Universidad Politécnica de Madrid

Escuela Técnica Superior de Ingenieros de Telecomunicación

A Personal Agent Architecture for Task Automation in the Web of Data. Bringing intelligence to everyday tasks.

Tesis Doctoral

Miguel Coronado Barrios
Ingeniero de Telecomunicación

2016

Universidad Politécnica de Madrid

Escuela Técnica Superior de Ingenieros de Telecomunicación

A Personal Agent Architecture for Task Automation in the Web of Data. Bringing intelligence to everyday tasks.

Tesis Doctoral

Miguel Coronado Barrios
Ingeniero de Telecomunicación

2016

Departamento de Ingeniería de Sistemas Telemáticos

Escuela Técnica Superior de Ingenieros de Telecomunicación

Universidad Politécnica de Madrid

A Personal Agent Architecture for Task Automation in the Web of Data. Bringing intelligence to everyday tasks.

Autor:

Miguel Coronado Barrios

Ingeniero de Telecomunicación

Tutor:

Carlos Ángel Iglesias Fernández

Doctor Ingeniero de Telecomunicación

2016

Tribunal nombrado por el Magfco. y Excmo. Sr. Rector de la Universidad Politécnica de Madrid, el día ________ de ____________ de ________.

Presidente: ______________________________

Vocal: ______________________________

Vocal: ______________________________

Vocal: ______________________________

Secretario: ______________________________

Suplente: ______________________________

Suplente: ______________________________

Realizado el acto de defensa y lectura de la Tesis el día ________ de ____________ de ________ en la E.T.S.I. Telecomunicación habiendo obtenido la calificación de ____.

EL PRESIDENTE LOS VOCALES

EL SECRETARIO

A mi familia,
en el ámplio sentido de la palabra.

Agradecimientos

Qué mejor que empezar esta memoria agradeciendo a todas aquellas personas que han estado a mi lado durante este viaje, y que han contribuído a que llegase a buen puerto.

Primero de todo, agradecer a Carlos su mentoría. Él es parte responsable de que esté satisfecho del resultado obtenido. Así mismo, no podría olvidarme de mis compañeros del "labo", Álvaro, "Jota", Oscar y tantos otros que han pasado y de los que he aprendido mucho. Sin olvidarme de todos aquellos ingenieros en ciernes, que con su trabajo han contribuido al resultado de esta investigación.

Igualmente, querría agradecer a Jürgen y Ralf el magnifico trato que recibí durante mi estancia en Hannover. Su empeño en que mi estancia fuese fructífera, también desde el punto de vista de la investigación, me ayudó a explorar opciones que de otra forma no hubiera considerado.

Por último, querría agradecir a mi familia, el que siempre estén ahí. Con la madurez te das cuenta de que su amor y apoyo incondicional es inestimable en los momentos en que más lo necesitas. Y te alegra profundamente el tenerlo. Y en especial agradecer mi niña, Olga, mi luz y mi alma. Que ha sufrido conmigo los altibajos de todo este camino. Ha sabido ser mi calma ante el agobio, mi sosiego ante el enfado y mi voluntad ante el desánimo. Muchas gracias.

Resumen

Internet está evolucionando hacia la conocida como *Live Web*. En esta nueva etapa en la evolución de Internet, se pone al servicio de los usuarios multitud de *streams* de datos sociales. Gracias a estas fuentes de datos, los usuarios han pasado de navegar por páginas web estáticas a interacturar con aplicaciones que ofrecen contenido personalizado, basada en sus preferencias. Cada usuario interactúa a diario con multiples aplicaciones que ofrecen notificaciones y alertas, en este sentido cada usuario es una fuente de eventos, y a menudo los usuarios se sienten desbordados y no son capaces de procesar toda esa información a la carta. Para lidiar con esta sobresaturación, han aparecido múltiples herramientas que automatizan las tareas más habituales, desde gestores de bandeja de entrada, gestores de alertas en redes sociales, a complejos CRMs o *smart-home hubs*. La contrapartida es que aunque ofrecen una solución a problemas comunes, no pueden adaptarse a las necesidades de cada usuario ofreciendo una solucion personalizada.

Los Servicios de Automatización de Tareas (TAS de sus siglas en inglés) entraron en escena a partir de 2012 para dar solución a esta liminación. Dada su semejanza, estos servicios también son considerados como un nuevo enfoque en la tecnología de *mash-ups* pero centra en el usuarios. Los usuarios de estas plataformas tienen la capacidad de interconectar servicios, sensores y otros aparatos con connexión a internet diseñando las automatizaciones que se ajustan a sus necesidades. La propuesta ha sido ámpliamante aceptada por los usuarios. Este hecho ha propiciado multitud de plataformas que ofrecen servicios TAS entren en escena.

Al ser un nuevo campo de investigación, esta tesis presenta las principales características de los TAS, describe sus componentes, e identifica las dimensiones fundamentales que los defines y permiten su clasificación. En este trabajo se acuña el termino Servicio de Automatización de Tareas (TAS) dando una descripción formal para estos servicios y sus componentes (llamados canales), y proporciona una arquitectura de referencia.

De igual forma, existe una falta de herramientas para describir servicios de automatización, y las reglas de automatización. A este respecto, esta tesis propone un modelo común que se concreta en la ontología EWE (Evented WEb Ontology). Este modelo permite comparar y equiparar canales y automatizaciones de distintos TASs, constituyendo un aporte considerable paraa la portabilidad de automatizaciones de usuarios entre plataformas. De igual manera, dado el carácter semántico del modelo, permite incluir en las automatizaciones elementos de fuentes externas sobre los que razonar, como es el caso de *Linked Open Data.*

Utilizando este modelo, se ha generado un *dataset* de canales y automatizaciones, con los datos obtenidos de algunos de los TAS existentes en el mercado. Como último paso hacia el lograr un modelo común para describir TAS, se ha desarrollado un algoritmo para aprender ontologías de forma automática a partir de los datos del dataset. De esta forma, se favorece el descubrimiento de nuevos canales, y se reduce el coste de mantenimiento del modelo, el cual se actualiza de forma semi-automática.

En conclusión, las principales contribuciones de esta tesis son: i) describir el estado del arte en automatización de tareas y acuñar el término Servicio de Automatización de Tareas, ii) desarrollar una ontología para el modelado de los componentes de TASs y automatizaciones, iii) poblar un dataset de datos de canales y automatizaciones, usado para desarrollar un algoritmo de aprendizaje automatico de ontologías, y iv) diseñar una arquitectura de agentes para la asistencia a usuarios en la creación de automatizaciones.

Palabras clave: Automatización de tareas, Web semántica, Social Stream, aprendizaje de ontologías, asistente personal, Agentes.

Abstract

The new stage in the evolution of the Web (the Live Web or Evented Web) puts lots of social data-streams at the service of users, who no longer browse static web pages but interact with applications that present them contextual and relevant experiences. Given that each user is a potential source of events, a typical user often gets overwhelmed. To deal with that huge amount of data, multiple automation tools have emerged, covering from simple social media managers or notification aggregators to complex CRMs or smart-home Hub/Apps. As a downside, they cannot tailor to the needs of every single user.

As a natural response to this downside, Task Automation Services broke in the Internet. They may be seen as a new model of mash-up technology for combining social streams, services and connected devices from an end-user perspective: end-users are empowered to connect those stream however they want, designing the automations they need. The numbers of those platforms that appeared early on shot up, and as a consequence the amount of platforms following this approach is growing fast.

Being a novel field, this thesis aims to shed light on it, presenting and exemplifying the main characteristics of Task Automation Services, describing their components, and identifying several dimensions to classify them. This thesis coins the term Task Automation Services (TAS) by providing a formal definition of them, their components (called *channels*), as well a TAS reference architecture.

There is also a lack of tools for describing automation services and automations rules. In this regard, this thesis proposes a theoretical common model of TAS and formalizes it as the EWE ontology This model enables to compare channels and automations from different TASs, which has a high impact in interoperability; and enhances automations providing a mechanism to reason over external sources such as Linked Open Data.

Based on this model, a dataset of components of TAS was built, harvesting data from the web sites of actual TASs. Going a step further towards this common model, an algorithm for categorizing them was designed, enabling their discovery across different TAS.

Thus, the main contributions of the thesis are: i) surveying the state of the art on task automation and coining the term Task Automation Service; ii) providing a semantic common model for describing TAS components and automations; iii) populating a categorized dataset of TAS components, used to learn ontologies of particular domains from the TAS perspective; and iv) designing an agent architecture for assisting users in setting up automations, that is aware of their context and acts in consequence.

Keywords: Task Automation, Semantic Web, Social Streams, Ontology Learning, Personal Assistant, Agent

Contents

CHAPTER 1

Introduction

Nowadays, users receive a huge amount of personalised information coming from social networks, sensor networks and smart phones. Processing all these streams is very time consuming. Users often get flooded with large number of notifications, and in most of the cases some pieces of relevant information pass unnoticed. Task Automation Services are a novel solution to this problem. They provide a mechanism for creating automations that react to these data without bothering users, thus saving them time and reducing information leaks. The research described in this thesis is centred on characterising these services, including their potential and limitations, and proposing how they can be improved thanks to the use of intelligent techniques.

This chapter introduces the motivation and the objectives of this Ph.D. thesis. Its goal is to show the reader what observations lead us to undertake the research developed as part of this work, the challenges we identified, and the contributions and objectives we pursued.

1.1 Motivation

The term *Live Web* (Windley, 2011a), also called the *real-time Web* (De Francisci Morales et al., 2012) or the *event Web* (Singh and Jain, 2010), describes a new stage in the evolution of the Web that extends the Web 2.0 or interactive Web. Instead of simply browsing static web pages or even interacting with a website, the Live Web is characterised by a brand new style of interaction through dynamic streams of information, called *personal streams*, to present contextual and relevant experiences to users (Windley, 2011a). There are several sources of personal streams such as: social networks, sensor networks, and mobile phones. These sources provide the necessary location-aware, relationship-aware, preference-aware and sensory context to achieve a new generation of context-aware applications (Beach et al., 2010).

Nevertheless, given that each user is a potential source of events, a typical user often gets flooded with many notifications; e.g. tweets, CRM notifications, chat messages, etcetera. As a result, they only read a small fraction of those they receive (De Francisci Morales et al., 2012), wasting tons of potentially useful data. Thus, several automation tools have emerged in order to simplify personal-stream management. Some of these tools are social media management tools (Kietzmann et al., 2011) such as: TweetDeck[1], with advanced filtering and scheduling facilities for Twitter; Rapportive[2], that combines Linkedin profiles with Gmail contacts; and many other data-driven mash-up tools (Yu et al., 2008), which provide users the ability to create new applications based on the available services.

What makes these tools really useful for consumers is that they combine incoming data available from different personal streams, presenting the information to the user in a manner that it is more useful than it was by separate. Thus, they create new personalised streams and services that behave in a particular way for each different user.

In this regard, a number of now prominent web sites, mobile applications, and desktop applications feature *rule-based task automation.* Typically, these services provide users the ability to define which action should be executed when some event is triggered, i.e. each user defines its own automations. Some examples of this simple task automation could be ‘When I am mentioned in Twitter, send me an email’, ‘When I come within 500 meters of this place, check-in in Foursquare’, or ‘Turn Bluetooth on when I leave work’. We call these services Task Automation Service (TAS) (Coronado and Iglesias, 2015a). Ifttt (Ifttt, 2015), Zapier (Zapier, 2015) or AutomateIt (Automateit, 2015) are a few enlightening examples.

[1]http://tweetdeck.twitter.com/
[2]http://rapportive.com/

This innovative technology offers a whole range of possibilities, e.g. to propose more complex and powerful automations, to gather data not only from web services but from physical devices, to auto discover these sources of data for convenience, or to distribute the execution of the automations that take advantages of the mobility of smart-phones, wearables and other connected devices. Moreover, being a novel field of research (Ifttt, the most relevant TAS, was founded on December 2014, less than one year prior to the time this research was started), most of these challenges are still open. Ultimately, it was a unique opportunity that was worth to take, to propose innovative solutions to some of these challenges, digging into the possibilities TASs offer.

1.2 Objectives

The primary objective of this thesis, is to build a personal agent architecture for task automation in the Web of data, bringing intelligence to the already mentioned Task Automation Systems. The vision of task automation as implemented by TASs, was born with them, and it is still emerging and at some points fuzzy. So it is necessary to set the foundations of this research establishing a framework to define and classify these services. Therefore, with that in mind, we have decomposed the thesis global objective into a number of more specific ones in order to advance towards the final solution step by step:

- **Compose a framework for classifying TASs**. Our objective is to study the existing TASs, soak in their functionalities and unique features, and shed light on the field building a framework that allows classifying and compare them. This goal involves to discover different paradigms that characterise TAS and their components, e.g. the privacy, the configurability, the communication, the availability, etc. the nature of components offered by the TAS studied, their execution profiles, As a result, we will obtain the sufficient understanding to i) give a formal definition of TASs, ii) describe and classify them and their components according to different criteria, iii) and devise a reference architecture for TAS that may fit any of the studied services.
- **Propose a semantic model for describing TASs**. Our objective is to identify the properties of the TAS components (channels, services, connected devices) used in automations to deliver a solution that would leverage this information to a common level. It would provide a common model for describing these components and their features, as well as automations created by consumers using these components. This model should allow comparing the component catalogue of different TAS, implementations of the same services given by different TASs, as well as the automations regardless

of the TAS used to create them. The formalisation of the model will benefit from advantages of semantic model, and should put impact on data interoperability and portability of automations, allowing external resource linking to the instances such as those from the Linked Open Data (LOD) cloud.

- **Discover vocabularies for specific services**. Our objective is to provide additional vocabularies, which extend the semantic model referred in the former step, with mappings to existing properties from well known vocabularies (e.g. Semantically-Interlinked Online Communities (SIOC), Friend of a Friend ontology (FOAF), Semantic Sensor Network ontology (SSN)). These vocabularies would leverage discoverability of components, and lift automation capabilities without harming descriptiveness of the original model. This goal involves the development of an algorithm for automatic ontology discovery that feeds from datasets of TAS component catalogues. Moreover, an automatic semantic extractor of TAS catalogues is developed, so that altogether with the algorithm the vocabularies are maintained up to date. Using semantic, linguistic, and structural information, manual assistance and guidance should be minimised if not removed.

- **Propose a personal agent architecture for task automation**. The last step consists in designing a personal agent TAS architecture that supports all features of TAS adding up the benefits of personal assistant, i.e. immediacy, proactivity, mobility (Myers et al., 2007). In that sense, proactivity refers to suggesting useful automations to each consumer based on their profile, location, or any other meaningful context data, even before they request them. It involves extending the general architecture proposed in the first step, to accommodate the agent module. The capabilities of this smart agent will benefit greatly from the semantic nature of the components and automations in order to perform reasoning and comparisons. This last step addresses the principal objective set at the beginning of this section, delivering a solution that fulfils it, therefore it closes this research work.

1.3 Structure of this dissertation

After introducing the motivation and objectives set for this thesis, the second chapter presents a framework for classifying TASs, surveys the most relevant ones –setting up the state of the art in that matter– and proposes a reference architecture for TASs. The second chapter helps the reader to have a clearer view of what TASs are, their components, their relation to similar approaches (such as mash-ups technologies), and the current state of commercial TASs as well as their evolution and trends.

Chapter 3 presents in detail an ontology named EWE for modelling TASs. It evaluates the coverage and accuracy of the ontology, and presents a brief use case scenario where the capabilities of EWE applied to TAS are unveiled. This chapter shows the reader the advantages of semantic reference models, how TASs benefit from them. Although it is difficult to retain it all in detail, the reader will have an intuition of the design principles of the ontology, and how it would be used to describe TASs and to build a semantic TAS.

Chapter 4 reinforces the semantic model describing a methodology that features automatic vocabulary discovery and learning for specific services in the task automation context. This methodology is capable of automatically i) extracting channel information from TASs' websites, ii) grouping the extracted channel instances in clusters based on the similarity of the events and actions they provide, and iii) generating vocabularies that complement EWE. This chapter shows the reader the motivation behind ontology learning approaches, and illustrates it with a methodology for ontology learning in the TAS context.

Chapter 5 describes a personal agent architecture for TASs that brings intelligence into task automations. It describes a message oriented middle-ware to be used to distribute messages in TASs, and particularises the reference architecture proposed in chapter 2 to a few use case scenarios, where the EWE ontology, the specific vocabularies and the message oriented middle-ware work together in conjunction with an agent system. This chapter helps the reader to understand the benefits of using agents in TASs, and to have a clear understanding of the possible tasks the agent develop to assist users. Finally, chapter 6 draws some conclusions, highlights the contributions, and proposes future research to continue this work.

This dissertation follows a timeline structure, since the order of the chapters meets the order the research was made. The outcome presented in each chapter is required in the next one. Consequently, it is also parallel to the order of the objectives set in the section 1.2.

CHAPTER 2

Task Automation Services: Automation for the masses

Task Automation Services have been gaining popularity in the past few years. However, the concepts and terms around TAS are still fuzzy, and there is not a frame to properly describe these services. Hence, this chapter surveys the most prominent TASs to extract the factors and features that define them. Those features are organised as dimensions, and outline a framework for classifying and comparing TASs. The chapter also defines a reference TAS architecture that identifies the key elements of TASs as well as their interactions. This architecture provides a common vocabulary and serves as a reference that can accelerate the development, adoption and evolution of TASs.

2.1 Introduction

Task automation permeates our daily lives, from the weather forecast that appears when the alarm clock rings, to the smartphone toast-notification that pops up every time we receive an incoming email. These automations orchestrate gadgets, Internet services and apps in a way that makes our life easier (Parks and Watkins, 2012). We are so accustomed to task automations, that sometimes it is hard to identify them, and even harder to realise that some years ago we used to perform those tasks manually.

While these predefined task automations are spreading across the web, a new user-centred fully-customisable approach is beating them all, the so-called task automation service (TAS). These services are typically web platforms or smartphone apps that provide a visual programming environment, where non-technical users can seamlessly create and manage their own personal automations (Meisel, 2014). The automation in these services takes the form of Event-Condition-Action rules that execute an action upon a certain triggering event i.e. "when triggering-event then do action". In the former examples, the alarm clock and the incoming email would be the triggers, whereas querying the weather forecast and displaying a notification are the respective actions.

Some TASs have become mainstream, such as Ifttt(Ifttt, 2015; Martinez, 2013) or AutomateIt(Automateit, 2015). In 2014, Ifttt reported more than 14 million web tasks created by end users. AutomateIt, an Android application, has more than 500,000 users. There are three success factors that explain their growing adoption. The first is usability; TASs provide a simple-yet-powerful intuitive interface for programming task automations. Hence, users experiment almost no learning curve when they start using them. The second factor is customisability; TASs let users program the automations they need. Although simple, automations are powerful improvements to users' daily lives. Giving users the capability to create their own rules awakens a sense of control and immediacy –they get the automations they need when they need them. The third factor is integration with existing Internet services. Users can automate tasks that access the Internet services they already use and are familiar with.

Given the novelty of Tasks Automation Services, this chapter aims to shed light on them. To better understand what TAS's automations might look like, consider the following scenario. Sarah uses a TAS every day, so she has defined a set of useful automations. Some automations notify her when something relevant to her happens, such as "when I'm mentioned on Twitter, send me an email" notify her when something relevant to her happens. Others, save her the bore of repeating a simple task, e.g. "when I'm tagged in a Facebook picture, save it to my Dropbox" or "convert incoming invoice emails to PDF and store them in my Evernote". Furthermore, Sarah also uses the smartphone app provided by the TAS. Once installed, the TAS can access several resources from her smartphone, so she can set up rules involving incoming calls, the camera, Bluetooth or GPS among others. Rules such as "when my smartphone's battery level is under 10 percent, text my parents" or "when I get to work, lower the volume of my ring-tone" take advantage of those capabilities. Moreover, the TASs feature Sarah enjoys most is the discovery of compatible services around her –using smartphone communication capabilities such as Bluetooth. This feature can automatically integrate her SmartTV or her home automation lighting system with the TAS, allowing her to set rules for home automation such as "when my alarm clock rings, switch on the bedroom lights".

This brief journey with Sarah illustrates how services and sensors can be connected by means of automation rules defined with a single TAS. It aims to give a clear view of the functioning and main features of TASs, and it also outlines some challenges, such as embracing smartphone resources or auto discovery of services; we will address these challenges in the following sections.

The rest of the chapter is structured as follows. First, Section 2.2 discusses the elements the former scenario introduced. Then, Section 2.3 defines a reference TAS architecture that identifies the key elements of TASs as well as their interactions. This architecture provides a common vocabulary and serves as a reference that can accelerate the development, adoption and evolution of TASs. Motivated by their relevance and popularity, Section 2.4 classifies the most prominent TASs according to different dimensions of the framework. Section 2.5 compares TAS to mash-up technology in order to measure how similar these two approaches are, and what are their key differences. Finally, Section 2.6 presents some concluding remarks of this chapter.

2.2 TAS Components: Channels and execution profiles

Our scenario combines events from Internet services, Sarah's smartphone, and connected devices. These services and devices are managed by channels. We define channels as abstractions for receiving events or emitting actions to Internet services (i.e. web channels) and connected devices (i.e. device channels). Channels should be registered in a channel directory service provided by the TAS. In this way, users can activate available channels when programming automations.

2.2.1 Web channels

Many TASs rely on third party Internet services to supply a pack of useful, popular, user-tested channels. Hence, users benefit from using TASs to manage the services they are already subscribed to (e.g. Evernote, Gmail). As a result, TASs provide users with a new layer of control to manage their services and they are not required to migrate.

By analysing the behaviour of web channels, we identify three characteristic dimensions. Consider a user that wants to define a new automation rule. First of all, that user needs to grant the TAS access to the Internet service, usually by providing access credentials – this is what *privacy paradigm* defines. In addition, some channels require to be configured. This is the case of the weather forecast channel, in which users need to provide a location for the forecast –this is defined within the *configuration paradigm.* Finally, channels may behave differently triggering the rule or being the consequence that takes place, i.e. they may generate events, provide actions or both –this is the *input-output (I/O) paradigm.*

From privacy paradigm's point of view, channels may be public or private. To activate a channel and let the platform act on behalf of users, they must grant access to the service. In our former example, Sarah had to allow the TAS to access her Dropbox account and email inbox to manage her files and emails. When this authentication is required, the channel is private. The privacy paradigm defines who will have access to events and actions provided by the channel. Information regarding private channels are for the user's eyes only, and it is tailored to the user. On the contrary, channels that do not ask for authentication are public channels, and every user gets the same information when using them. This is the case of news feeds or weather forecast channels. The privacy paradigm also covers private group channels, where every group member receives all the events the channel generates. These channels are common in scenarios like home automation, where family members are likely to share home channels.

The configuration paradigm defines the set-up needed to activate a channel –apart from authentication. Public channels usually require configuration for the sake of better user experience and also as a matter of efficiency. For instance, in our example to activate a weather channel Sarah provided the location where she lives. Hence, she will receive weather events related to that location. In general, private channels don't require configuration since they are already tailored to the user.

Finally, when activated, channels might generate events, provide actions or both. This is what the input-output paradigm defines. Events are changes in the service state, e.g a new email on Sarah's inbox. On the other hand, channels might also offer action capabilities, e.g. switching on the bedroom light. IO paradigm also covers pipe channels: those that process the input to generate a different output, that can be wired to another channel, For instance, the PDF converter channel that saves the content of Sarah's email into a PDF file is then connected to the Evernote channel to store the file.

From an integration perspective, most of the efforts in offering a new web channel are related to implementing the protocol to communicate with the Internet service behind the channel. This is TAS administrators' duty, which depends on the availability of an API for the Internet service. The authorisation process involved in accessing the Internet service API determines the privacy paradigm. Besides communication with the service, the TAS administrators define what events and actions will be offered as part of the IO paradigm as well as the configuration paradigm.

2.2.2 Device channels

In comparison to Web channels, device channels manage data from the connected devices they manage. In Sarah's scenario, her home automation lighting system and the Smart TV provide device channels to control all the switches of her home and her Smart TV, respectively.

From a behavioural approach, device channels respond to the same three dimensions analysed for web channels. In addition, device channels implement two additional dimensions, the communication paradigm and the discovery paradigm. The *communication paradigm* defines how the communication between the devices and the channel will be carried out: wired, wireless, through the Internet, etc. It also defines on what conditions the channel is available. As opposed to web channels, which may be accessed from all around the world through Internet connection, access to device channels may depend on local aspects. These aspects are part of the communication paradigm. For instance, when using wireless communication, availability is subject to the distance between devices, i.e. being under coverage area or not. Finally, device channels may announce themselves so that they are available for automations as in Sarah's scenario. This is what the *discovery paradigm* defines. It provides standard operations and APIs to enable self-identification of devices, capabilities discovery, and access to device data using pre-defined message structures. As a result, the system provides a "plug and play" capability.

From the point of view of implementation, each TAS administrator decides which communication protocols will be supported in the platform. Each protocol has its own restrictions about range, power consumption, number of connected devices, etc.. They also provide mechanisms such as security and device discovery. To communicate two devices, they need to support the same protocol. However, this should not be an issue, as a device may implement several protocols. In fact, many devices are compatible with the most widespread protocols, e.g. WiFi, ZigBee, Z-Wave, even Bluetooth (Labiod et al., 2007).

2.2.3 Rule execution profiles

This section describes automation rules, which provide the logic to connect channels. As previously stated, TASs automations address simple Event–Condition–Action (ECA) rules (Beer et al., 2003). The rule's event and action may come from the same or different channels. However, more complex rules could be devised: *multi-action* rules can execute several actions in parallel when the rule is executed; *multi-event* rules are triggered by a combination of events; and chain rules execute a list of actions in sequential order, so the output of an action may be used to trigger the next rule. Complex rules require the TAS to support additional features. For instance, multi-event rules require Complex Event Processing (CEP) support to evaluate complex patterns of events, and chain rules make use of pipe channels, so they must be supported by the TAS.

Group rules are a particular kind of rules that involve several users and are susceptible to collisions with other rules. For instance, Sarah's Smart lighting system is a shared resource managed by a group channel. If Sarah defines a rule to switch off the corridor lights while she is asleep, and her flatmate had a rule that turns them on when the alarm clock rings, both rules collide. It is easy to get to a point where the TAS cannot determine if the lights should be on or off. As a rule of thumb, rules that include group channels are group rules.

The rule *execution profile* defines where the execution of the rule is taking place. Rules may be executed according to different execution profiles to increase performance and enable off-line rule execution. In Sarah's scenario, she uses a TAS hosted in the web, but we can imagine other scenarios where a smartphone or a set-top-box performs the automation. Rule execution may be accomplished according to three different execution profiles: entirely on the web, on the mobile client, and mixed execution. A *Web-driven execution profile* centralises the execution on the server, allowing the existence of lightweight clients at the cost of requiring Internet connection. A *mobile-driven execution profile* is orchestrated on the client (e.g. smartphone, set-top-box), allowing off-line rule execution when only local device channels are involved. A *mixed execution profile* takes the advantages of both profiles. It can shift the execution to the client or to the Web, which also reduces communication payload between client and server.

2.3 A Reference Task Automation Service Architecture

Once we have described the main components of TASs, we define Task Automation Service as a service that lets users create automation rules that connect channels using a visual editor. A reference TAS architecture must incorporate the features and components previously described (channels, rules, execution profiles) and also provide support for some of the challenges presented in Sarah's scenario. The following are the architecture requirements:

- provide a visual rule editor for creating rules;
- include both Web and device channels;
- feature channel discovery, providing adapters for connected devices so that they are reachable directly by the platform;
- enable multi-event, multi-action and chain rules;
- manage group channels and group rules;
- detect collisions with rules that involve group channels; and
- support a mixed-execution profile.

The architecture must provide a visual automation rule editor that users could use to create their automations, but also to activate channels according to the privacy paradigm. To provide access to Web and device channels, the architecture must provide the logic needed to connect with Internet services and connected devices. Moreover, the TAS must be notified without delay when an event is generated by the channels, and it must be able to send actions. It also must be able to discover channels according to the discovery paradigm of device channels.

Advanced rule features such as multi-action rules and chained rules do not imply additional requirements, because they can be translated in a set of simpler rules. Multi-event rules involve temporal reasoning of events, and so the TAS requires CEP facilities (Eckert et al., 2011). Nevertheless, a trade-off among usability for end users and expressiveness should be reached for multi-event rules. Group rules require group channels support and rule collision handling. To handle collisions, the architecture must be able to first detect them, and then act when the collision occurs and prevent any unwanted effects.

Supporting mixed-execution profile requires some additional logic to coordinate server's and device's rule engines while they orchestrate rule execution. The logic is also needed to

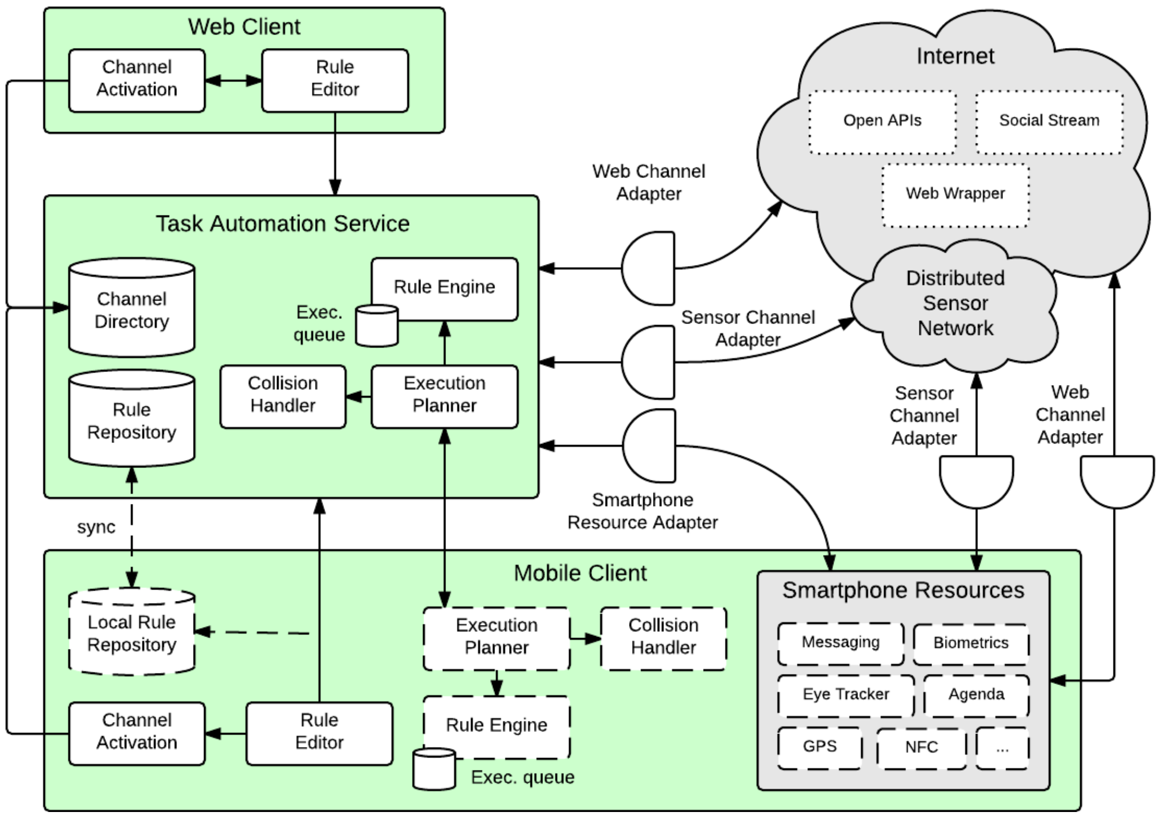


Figure 2.1: Reference Task Automation Service Architecture general diagram. Elements in dashed lines are optional in some implementations.

guarantee that the information about the user and the rules are synchronised on the device and the server.

Once the requirements to support these novel features are clear, we introduce a reference architecture shown in Figure 2.1 which fulfils them.

Rules may be created using an editor on a web client or a mobile client. They are stored in a central rule repository on the TAS server. However, since those rules that can be executed on the client according to the mobile driven execution profile, they are synchronised with a local rule repository for off-line access. Mobile and web clients also allow users to activate channels. Once a channel is activated, it is saved in the channel directory together with the authorisation credentials. These credentials are used by the adapters to access the channels.

Adapters provide uniform access to all kinds of devices. They are responsible for notifying the TAS of incoming events, commanding the execution of actions and taking charge of channel authentication. Adapters can be implemented following a publish-subscribe (Eugster et al., 2003) or polling strategy to get notified of device events depending on their nature.

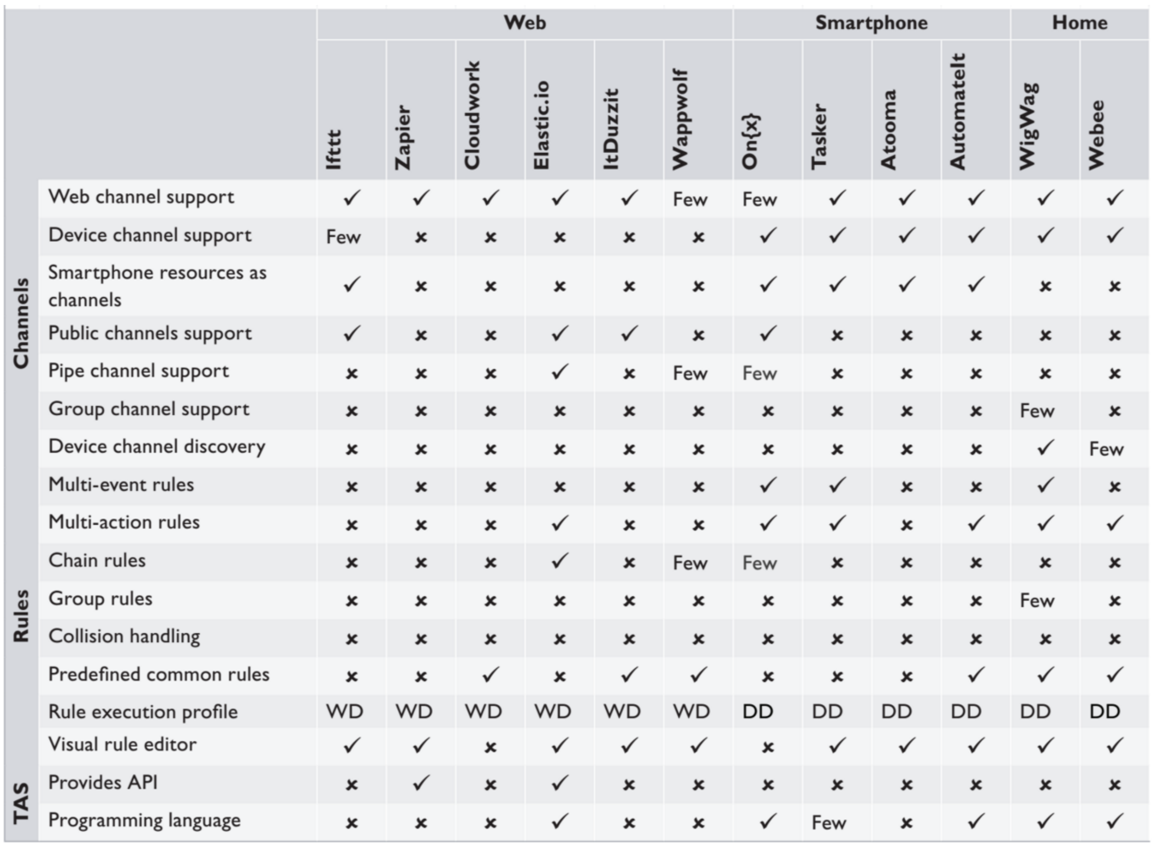
The implementation of adapters can be done by the TAS administrator or by third parties if the TAS provides an adapter SDK. Adapters to sensor channels provide two different paths: access through a web-protocol, or direct access using access protocols such as ZigBee, Z-wave, Bluetooth or WiFi. They usually expose a set of sensors, i.e. a sensor network, but single device channels are also feasible.

To support a mixed-execution profile modules involved in rule execution and channel access must work in coordination. This is the case of the rule engine responsible for executing rules and managing rule life cycles within the execution query. Rule execution consists in fetching the incoming triggering event, extracting the arguments and using those parameters to request the action execution. The Execution Planner guides the orchestration from a higher level according to the active rule execution profile. It manages the state of the channels (within the channel directory), tracking when channels are down and new channels are discovered. For instance, when Sarah arrives at home the Smart TV channel is discovered and added to the channel directory. In turn, when she leaves home, the channel will be down because it is out of range.

Finally, the collision handler analyses the rules in the repository, searching for possible collisions among them. Some patterns of collision are easy to detect e.g. the simplest collision consists of two rules with the same triggering event that try to execute two opposite actions. It takes into consideration which users are currently connected to each channel (as registered in the channel directory), since rules of two users can only collide if they both are connected to the same channel. Recall the example where Sarah wants to have the corridor lights off while she is asleep, and her flatmate set a rule to switch them on when the alarm clock rings. If Sarah's flatmate is not at home, there is no possibility of collision because the channel is not active for Sarah's flatmate. The execution planner consults the collision handler before executing a rule, and in case that rules collides with other rules, its execution is skipped.

2.4 Analysis of current TAS platforms

To determine which of the features discussed are supported by state of the art task automation, we have analysed web platforms for general audience (Ifttt (Ifttt, 2015)), web platforms for business and enterprises (Zapier (Zapier, 2015), Cloudwork (Cloudwork, 2015), elastic.io (Elasticio, 2015), itduzzit (itduzzit, 2015)), a web platform for cloud storage synchronisation (Wappwolf (Wappwolf, 2015)), mobile apps (Tasker (Tasker, 2015), Atooma (Atooma, 2015), Automateit (Automateit, 2015), onx (Onx, 2015)), and smart home systems (WigWag (WigWag, 2015), Webee (Webee, 2015)). A summary of the results is presented in



		Web						Smartphone				Home	
		Ifttt	Zapier	Cloudwork	Elastic.io	ItDuzzit	Wappwolf	On{x}	Tasker	Atooma	AutomateIt	WigWag	Webee
Channels	Web channel support	✓	✓	✓	✓	✓	Few	Few	✓	✓	✓	✓	✓
	Device channel support	Few	×	×	×	×	×	✓	✓	✓	✓	✓	✓
	Smartphone resources as channels	✓	×	×	×	×	×	✓	✓	✓	✓	×	×
	Public channels support	✓	×	×	✓	✓	×	✓	×	×	×	×	×
	Pipe channel support	×	×	×	✓	×	Few	Few	×	×	×	×	×
	Group channel support	×	×	×	×	×	×	×	×	×	×	Few	×
	Device channel discovery	×	×	×	×	×	×	×	×	×	×	✓	Few
Rules	Multi-event rules	×	×	×	×	×	×	✓	✓	×	×	✓	×
	Multi-action rules	×	×	×	✓	×	×	✓	✓	×	✓	✓	✓
	Chain rules	×	×	×	✓	×	Few	Few	×	×	×	×	×
	Group rules	×	×	×	×	×	×	×	×	×	×	Few	×
	Collision handling	×	×	×	×	×	×	×	×	×	×	×	×
	Predefined common rules	×	×	✓	×	✓	✓	×	×	×	✓	✓	✓
	Rule execution profile	WD	WD	WD	WD	WD	WD	DD	DD	DD	DD	DD	DD
TAS	Visual rule editor	✓	✓	×	✓	✓	✓	×	✓	✓	✓	✓	✓
	Provides API	×	✓	×	✓	×	×	×	×	×	×	×	×
	Programming language	×	×	×	✓	×	×	✓	Few	×	✓	✓	✓

* ✓= supported; ×= not supported; Few = few support; WD = Web-driven execution profile; DD = device-driven execution profile;

Figure 2.2: Reference Task Automation Service Architecture general diagram.

Figure 2.2 and the complete report is available online (Coronado and Iglesias, 2015c).

As expected, web channel support is much larger than device channel support. Although the studied apps manage the resources in the smartphone, only home automation related TASs provide device channels that connect directly to devices. Pipe channels are only fully-supported by elastic.io. It is a powerful concept, and it integrates perfectly in their interface; however, it is barely used. It is worth mentioning that home automation TASs feature channel discovery and group channels. Unfortunately, their support is still extremely limited.

Regarding different types of rules, few TASs support multi-event rules' complexity in comparison to simpler rules. Multi-action rules have wider support, but still many TAS managers do not include them in their platform in order to keep rule editors simple. In the end, a user may achieve the same functionality by implementing a rule for each action in the multi-action rule. Elastic.io, which supports pipe channels, features chain rules, and wigwag and webee, that support group channels, feature group rules –however, none of them handles collisions in group rules. Finally, some TASs provide their users with a pack of predefined rules that proved to be useful for previous users, because they act as a shortcut for creating those automations.

At the time we performed this study, none of these platforms supported a mixed-execution profile. Thus, we split them into those with a web-driven execution profile (Ifttt, Zapier, Cloudwork, elastic.io, itduzzit and wappwolf) and those with a device-driven execution profile (Tasker, Atooma, Automateit, onx, wigwag and webee). By their nature, platforms with a Web-driven execution profile have more limited access to device channels than those executed on the smartphone. The latter has access to all the smartphone resources. For that reason, Ifttt has already released a smartphone app that grants Ifttt server access to smartphone resources. In turn, Zapier includes Tasker as a channel that effectively grants access to those channels too.

Several TASs offer advanced features for users with programming skills to set up automations using a programming language. This is the case of Onx (javascript), AutomateIt (bash), elastic.io (javascript), Itduzzit (proprietary), wigwag (Arduino/Raspberry/-javascript) and webee (boss). Moreover, Zapier, elastic.io, Tasker and webee provide an API for developing channels (some call them plugins).

Apart from the technical aspects and features considered in the framework, there are other characteristics that may differentiate them from each other. The most important is target audience, which defines the type of channels the TAS integrates. Ifttt, AutomateIt and Atooma focus on the general audience, integrating the most popular channels. For this audience segment, automation of smartphone resources is essential, because they use these devices much often than their computers. This is why AutomateIt and Atooma are smartphone apps, and Ifttt developed its own smartphone app. Zapier, Cloudwork, and Elastic.io target business users. They integrate wide variety of specific channels (e.g. CRM, project managing tools, etc.) aiming to support as many services as possible, so that they cover all business needs. Tasker and onx aim at more technical users, supporting more complex rules. And obviously, Wigwag and Webee being smart home hubs, give support to smart homes and IoT integration.

The target audience is in certain correlation with the pricing policy, which includes freemium service, monthly fee, one payment, upgrade purchases, etc. TASs for business customers choose monthly fee policy, while general audience TASs opt for freemium models; smartphone apps usually include in-app purchases, except for Tasker which is not a free app; and smart home hubs integrate TAS functionalities so buying customers pay for the device.

Concept / name	Ifttt	Zapier	Kinetic	Triggers	Tasker	On{x}
Rule	Recipe	Zap	Rule	Trigger	Profile	Rule
Channel	Channel	App	Endpoint	–	– (plugin)	Object
Event	Trigger	Trigger	Event	Input	Context	Trigger
Action	Action	Action	Action	Output	Act./Task	Action

Table 2.1: Disparity in TAS Nomenclature.

To promote their product and favour user activation, some TASs are building a community around their users automations. Ifttt, Zapier, and AutomateIt provide a *gallery* of automations were users may inspect other's automations, rate and favourite them, and easily add them to their automation portfolio[1] in a one-click operation. This benefits user engaging, helps users to discover useful automations, and the system may use their profiles for better rule recommendation. In particular, Ifttt is watching over this section –they even include a *gamification system* to reward the users that create the most popular automations.

Furthermore, while analysing these TASs, we noticed there is a lack of agreement in the name given to the same concepts. The Table 2.1 presents some examples of different terms used to define the same elements. This fact shows the concept TAS as a type of service among other productivity tools is not fully developed yet. It makes harder for users to figure out how each TAS works, and also for the research community and developers to understand the design, and even the documentation. In the following chapter we address this issue and propose a solution to overcome it.

Apart from commercial systems, there are several approaches to TASs in the specialised literature. Related to the Web of Things (WoT), Paraimpu (Pintus et al., 2012) allows user to interconnect Http-enabled smartobjects and web services. In the terms we use, it is a TAS that supports configuring sensors and devices as data streams. Among the current TASs described before, several of them support using physical sensors and actuators from different approaches. Ifttt, includes channels to handle sensors such as WeMo[2] or PhillipsHue[3]. Its approach bypasses communication to them using third party APIs. SmartThings or Webee directly support managing sensors and but only those they provide. In this regard, Parimpu is more flexible than any of them since it allows configuring your own smartobjects as chan-

[1]The set of personal automation orchestrated by a user
[2]http://www.wemothat.com/
[3]http://www2.meethue.com/

nels. However, final users need programming skills to configure them. As Karger (Karger, 2014) points out, one of the main disadvantages of these systems is that it is impossible to integrate new data suppliers and consumers unless the TAS company chooses to do so. To overcome this problem, Opasjumruskit et al. designed Mercury (Opasjumruskit et al., 2012), a powerful TAS that features service discovery. This service is able to find appropriate sensors, services, actuators, etcetera; to perform certain functionality. Mercury relies on semantic annotation of web of device data sources so it can reason about them.

Regarding better user interfaces, Atomate it (Van Kleek et al., 2010) focuses on providing a constrained-input natural language interface for composing automations in natural language. Although it is not an entirely free-text approach, it is very intuitive and the authors claim that users increase their satisfaction compared to the standard interfaces of commercial TASs.

2.5 TAS and web mash-ups

Some researchers consider that TASs borrow inspiration from Web Mash-ups; i.e. applications generated by combining content, presentation, or application functionality from disparate Web sources (Vladimir et al., 2015). This is because TASs also compose services and combine data from different web data sources. After analysing the existing TASs, we identify five dimensions in which they may be compared to web mash-ups. The first dimension is the *number channels or widgets*, defined as the number of sources or elements that are available to be used in the compositions. It depends on the number of data sources that are integrated in the mash-up engine or the TAS. In both cases, it depends on the developers that support the system, so their performance in this dimension is similar. The second dimension is the power or the automations, or the resulting mash-up; i.e. the *capabilities* offered to users. Mash-ups lead this field since they offer data visualisation, filtering, and processing to name a few. Next, the *ease of use* is also considered; i.e. how easy is to configure the automations or build the mash-ups for the user. In this dimension, mash-ups usually involve complex data pipelining when not programming skills. On the other hand, TASs are characterised by presenting their users intuitive interfaces to build their automation. The ease of use in one the upsides that made TASs so popular these days, since almost all Internet user is a potential TAS user. Customisability or *personalisation* is another advantage of TASs. Each user is able to compose their own task automations, as opposed to mash-ups, that are packed serviced delivered to the user. More importantly, TASs allow users to utilise their own personal streams. However, since TASs are focused on being as easy to use as possible, some customisation capabilities are skipped. Finally, *portability* is the fifth dimension. This

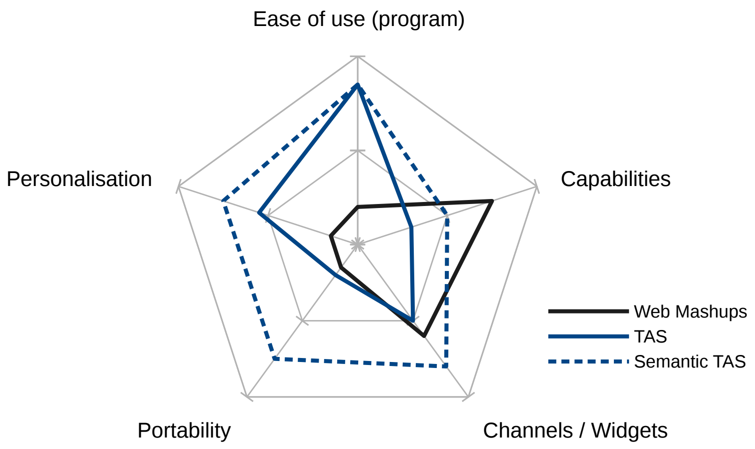


Figure 2.3: Comparison between mash-ups and rule based TASs.

is the capability of exporting and importing automations/mash-ups to different engines or systems. Both, TASs and mash-ups perform very low in this dimension because typically there is not any support for this.

After a qualitative analysis based on experts insights, all these dimensions are compiled in the chart shown in Figure 2.3. According to the given explanation, the reason of the higher penetration of TASs compared to mash-ups is their personalisation and ease of use. There is a third kind of system considered in the chart: semantic TASs, i.e. TASs which employ semantic technologies. These provide several advantages over classic TASs in most of the dimensions shown in the chart. In the following chapter, we explore the benefits of semantic TASs and propose a solution to implement them.

2.6 Discussion and Outlook

As TASs gain in popularity and existing TASs increase their number of users, new services appear and compete. As a novel domain, task automation opens the door to several opportunities –yet many challenges remain. First, a certain degree of standardisation is required. Because each TAS wages war on its own, Internet service developers might find their services available as channels in some TASs, but not in all of them. Moreover, when users define rules within the scope of a TAS, these rules can not be exported to others, nor shared. Second, TASs should evolve into a mixed execution profile that is able to interact with Web and device channels. Third, TASs that include device channels to manage shared devices (such as a smart TV or a smart lighting system) should allow group rules and consider the mechanism for detecting and handling collisions. This is quite challenging and will require complex algorithms. Finally, TASs should include mechanisms for discovering device channels so that their configuration is almost transparent to the end user.

This chapter offers an overview of *Task Automation Services* (TASs); their features, and to what they owe their popularity. The chapter surveys the most important commercial platforms, mobile apps and smart home hubs that feature task automation. This survey is summarised in Figure 2.2. Furthermore, an extended report is available online (Coronado and Iglesias, 2015c) as additional material of this chapter.

This survey arises several conclusions. Firstly, TASs that focus on different user audience implement different feature sets; e.g. Ifttt (Ifttt, 2015) or AutomateIt (Automateit, 2015), that target the general audience, offer neat and easy to use interfaces with connectivity to smartphone channels. On the other hand, Zapier (Zapier, 2015), Elastic.io (Elasticio, 2015) Cloudwork (Cloudwork, 2015), that target the business users, pay more attention to the power and capabilities of the rules. Secondly, there is a lack of agreement in the nomenclature used to refer to the same concepts, Table 2.1 gathers these variations. Thirdly, the TASs studied suffer from two major shortcomings, they lack of: support for channel discovery; and, reasoning over contextual data and *Linked Open Data*(LOD).

Since several authors consider TASs an evolution of web mash-ups, this chapter also describes a framework embracing five dimensions to compare them. The conclusion of this study is that TASs perform better in the personalisation and easy of use dimensions; while web mash-ups feature better automation capabilities. In the comparison, we also include TASs with semantic capabilities, as the prototype introduced in Section 3.4 which improves TASs performance in all five dimensions.

Finally, we identified three main trends in the market. First, web-based TASs are developing smartphone apps in order to integrate smartphone resources as channels of their catalogue. Second, existing Web TASs are starting to offer Rest APIs in order to delegate channel integration to Internet service developers. Thus, Internet service developers interested in integrating their services as channels will have a way to do it themselves. Finally, home automation systems are acquiring this vision, incorporating custom rule automations in their hubs, moving from ‘remote control’ to ‘automate control’.

CHAPTER 3

Modelling Rules for Task Automation: the Evented Web Ontology (EWE)

This chapter presents a reference model for describing TASs. Having identified there is a need of standardisation in the domain, a reference model can fulfil this need and provide several additional advantages. The proposed model is implemented as an ontology named EWE, which is compiled following a formal procedure for extracting the main concepts and relations of the most relevant TASs studied. Finally, its capability for describing TASs has been evaluated and a use case scenario illustrates its functionality.

3.1 Introduction

The novel vision of task automation as implemented by TASs is at an early stage of development, in fact some concepts involved are still fuzzy. For instance, the lack of consensus in naming these concepts (as shown in table 2.1, in the former chapter) is an evidence of that. This disparity makes harder to communicate ideas, to discuss proposals, or to compare different systems.

Reference models stand as a solution to this issue, setting a framework for communicating concepts at a particular environment. According to Mackenzie et al. *"A reference model is an abstract framework for understanding significant relationships among the entities of some environment"* (MacKenzie et al., 2006). Its purpose is *"to provide a common conceptual framework that can be used consistently across and between different implementations and is of particular use in modelling specific solutions."* Although the methodical description of the TAS's components shown in the former chapter constitutes a framework to be used for comparing TAS's channels, rules and execution profiles, there is not a formal reference model that comprises concepts, relationships and entities. Thus, reference models make easier for TAS managers to communicate to their users and to the research community, for developers to be aligned to meet the requirements of the architecture, and allow users to easily understand where these products fit into their needs.

One of the most common implementations to reference models are the ontologies. Defined by Gruber (1993) as explicit specification of a conceptualisation, they enhance reference models with the benefits from the Semantic Web (Berners-Lee et al., 2001). There exists ontologies for almost any domain of knowledge. Among the most popular ontologies and vocabularies are the SIOC ontology for modelling online communities, the GeoNames Ontology (GN) for adding geospatial semantic information to the Word Wide Web, or the vCard vocabulary for describing contact information as in business cards.

Regarding the task automation domain, semantic models offer five major advantages apart from those inherited from reference models. With a common semantic model for describing channels and automations, i) users can export their automation portfolio and load it on a different TAS since both work with the same model. ii) Semantic description of automations enables reasoning over external resources. So, a semantic rule engine with access to the LOD Cloud, could trigger automations with complex conditions that depend on external resources. Somehow, the LOD becomes a new public channel to the users (as given by the privacy paradigm described in Chapter 2). iii) External software agents can read, understand and thus reason with the data from TAS. This enables other services or programs to be developed around the TASs, enhancing the scenario. Apart from these agents, iv) it facilitates the entrance of new actors –whether they are new TASs or channel providers– because of compatibility reasons, and ease of integration. Finally, v) it also facilitates the analysis of domain data: the extraction, processing and analysis.

In this scenario, we present a threefold contribution. The first contribution of this chapter is the development of an ontology called the Evented WEb ontology (EWE) to model TASs. This ontology was designed to offer all the benefits recently described, and it was compiled using a formal iterative process of feature extraction. It also unifies the several dimensions of TAS components presented in Section 2.2 in the former chapter. The second contribution is the proposal of a use case scenario to show de advantages of EWE. Within that use case, we developed a prototype of a semantic TAS using EWE. Finally, the third contribution is an ontology evaluation that validates that EWE covers the domain it models by comparing its coverage and accuracy to popular alternative approaches. The rest of the chapter is organised as follows. First Section 3.2 describes the background in several topics that are addressed throughout the chapter. Next, Section 3.3 presents the EWE ontology: its design methodology, the main classes, properties, mapping to existing external ontologies, and examples of EWE usage. An implementation of a semantic TAS featuring EWE and a use case are presented in Section 3.4. Then, in Section 3.5, EWE is formally evaluated. Finally, Section 3.6 concludes and gives future works.

3.2 Background

In this section, we introduce some of the background technologies and efforts made in the state of the art that are related to the research described in this chapter. It includes reference and semantic models, representation of rules and some scientific approaches to task automation services that cover some challenges identified in the former chapter.

3.2.1 Reference and semantic models

An ontology is an explicit specification of a conceptualisation (Gruber, 1993). As implementation of reference models using Semantic Web technologies, ontologies offer several advantages that justify their usage. One of the most common reasons for developing ontologies is sharing common understanding of the structure of information among people or software agents (Musen, 1992). Probably, the most representative example to illustrate it is the SIOC ontology, which provides the main concepts to describe online communities (Breslin et al., 2005). When blogs and news sites use the SIOC ontology, software agents are able to extract and aggregate information from these different sites. It allows the agents to easily perform operations such as compiling a list of articles of an author, combining posts about the same event, or creating a timeline of all the entries of a certain topic. Other ontologies like the Music Ontology (Raimond et al., 2007, 2013) for describing music (artist, album, tracks, etc.) or the Good Relations ontology (Hepp, 2011) for describing e-commerce stores and products, provide the same advantages in similar scenarios.

As with the linked-data initiative, ontologies enable reuse of domain knowledge. Best practices in ontology development encourage to create mappings to concepts of existing ontologies, to avoid concept redefinition. There are several sites collecting Resource Description Framework (RDF) vocabularies and Ontology Web Language (OWL) ontologies. FOAF ontology (Brickley and Miller, 2014), the Dublin Core (dcterms), or Simple Knowledge Organization System (SKOS) are among the most reused ontologies (Jentzsch et al., 2011). In this regard, there are several efforts aiming for automatic ontology mapping discovery. Jain et al. (2011) propose a mechanism to aid ontology developers to find schema-level links between two LOD ontologies. Nikolov and Motta (2010) assist in discovering relevant repositories for interlinking and comparing them with respect to the coverage of specific domains.

Another common use of ontologies is separating the domain knowledge from the operational knowledge. We can describe a task of configuring a product from its components according to a required specification and implement a program that does this configuration independent of the products and components themselves (McGuinness and Wright 1998). We can then develop an ontology of PC-components and characteristics and apply the algorithm to configure made-to-order PCs. We can also use the same algorithm to configure elevators if we “feed” an elevator component ontology to it (Rothenfluh et al. 1996).

Analysing domain knowledge is possible once a declarative specification of the terms is available. Formal analysis of terms is extremely valuable when both attempting to reuse existing ontologies and extending them (McGuinness et al. 2000).

Apart from the ontologies mentioned to illustrate the advantages of ontology development, there are many other ontologies modelling many domains. In the context of TAS, the range of suitable mappings is very wide, because supported channels may represent services of very different nature. For instance, the Description Of A Project (DOAP) ontology may be used to describe projects used by channels like Trello or Github, the Tag Ontology (TAGS) ontology for describing bookmark from services like Evernote or Bitly, or the SSN for describing sensors.

3.2.2 Rule based event processing

In TAS architecture, adapters act as data-streams driven by events, they trigger data generation that is then distributed to all the subscribers. Those events are usually combined to each other producing more complex ones that enhance event selection according to the context of the user. Therefore, there is a need to process those events efficiently and concurrently so CEP technologies arise.

Several standardisation attempts have been made in rule based event processing and complex event processing. A thorough overview of these efforts and their limitations can be found in the work by Paschke et al. (2011). Among them, some notable approaches for event vocabulary standardisation are the WSDM Event Format (WEF) standardised by OASIS (Bullard et al., 2006) and the semantic XML format of AMIT that includes temporal extensions (Aberer et al., 2007). Event Processing Technology Society (EPTS) Reference Architecture (Paschke and Vincent, 2009) defines an initial functional reference architecture describing the typical event processing and complex event processing operations.

Research shows much overlap among existing standards for business and software at multiple levels (Paschke et al., 2011). Shaw et al. (2009) provide a comparison of existing event ontologies, nevertheless, as pointed out by Paschke et al. (2011), none of these ontologies have been developed specifically for CEP or being used in combination with CEP technologies. Westermann and Jain (2007) give a thorough view of how events are used in multimedia systems –describing some examples of traditional and emerging uses– ranging from programming frameworks to *life logs* and *the Event Web*. They provide a catalogue of requirements, and sketch a multimedia event model that addresses those requirements. Gu et al. (2005) model events as the outcome produced by sensors, and use them to trigger context-aware rules and services. Ricquebourg at al. (Ricquebourg et al., 2007) applied the same vision at a Smart Home scenario using Semantic Web Rule Language (SWRL) rules.

3.2.3 Automations as Event-Condition-Action rules

According to the classification by Boyle et al. 2007, rules can be classified into three categories: *deduction rules*, *normative rules* and *reactive rules*. Deductive rules are logical statements for deriving new knowledge from other knowledge by using logical inference. Normative or consistency rules define constraints on the data or business logic in order to ensure data consistency. Finally, reactive rules describe reactive behaviour and usually follow the form of ECA rules, that is, *ON Event IF Condition DO Action*. This is the type of rules that better represent the automations orchestrated in TASs.

The ECA rules paradigm has been widely used in many application fields, ranging from active database management systems (Paschke and Kozlenkov, 2009), workflow management systems (Bae et al., 2004; Cugola and Margara, 2012), ambient intelligence (Augusto and Nugent, 2004; Sadri, 2011) to service integration (Ipiña, 2001).

In the service scenario, several approaches have been followed for monitoring the composition of web services. A large body of research has been based on assertions. ECA rules have been used for the definition of policies, which has been standardised in WS-Policy. An interesting extension is WS-Policy4MAS (Erradi et al., 2006) that enables the specification of policies for monitoring of functional and QoS aspects. Other approaches have monitored QoS statistics based on CEP techniques (Moser et al., 2008, 2010) or temporal properties of web services compositions (Barbon et al., 2006).

3.2.4 Rule representation in the Semantic Web

OMG Production Rule Representation (PRR) defines a rule meta-model and profiles for production rules in order to provide interoperability across modelling tools and inference engines. Along with other types of rules, the specification considers the need for defining an ECA profile based on this meta-model but this work is still an open issue (OMG, 2009).

One of the initiatives involved in PRR model is RuleML (Boley et al., 2010; rul, 2010), a non profit specification with the aim of facilitate sharing and publishing rules in the web. The first specification of the standard covers deliberation and reaction rules, and provides a concrete syntax to PRR. The branch of reaction rules includes ECA and CEP rules. RuleML rules can combine all parts of both deliberation and reaction rules, which enables uniform XML serialisation across all kinds of rules.

SWRL (O'Connor et al., 2005; SWRL, 2005) takes parts from OWL CL and OWL Lite, and extends the set of OWL axioms to include Horn-like rules (which may be used in ECA rules). The proposed rules are of the form of an implication between an antecedent (body) and consequent (head). The intended meaning can be read as: whenever the conditions specified in the antecedent hold, then the conditions specified in the consequent must also hold. Thus, SWRL is able to describe ECA rules. The XML Concrete Syntax is a combination of the OWL XML Presentation Syntax with the RuleML XML syntax. It has several advantages regarding the Semantic Web: i) rules are explicitly defined in RDF format; ii) arbitrary OWL classes (e.g., descriptions) can be used as predicates in rules; iii) rules and ontology axioms can be freely mixed; iv) interoperability between OWL and RuleML is simplified, existing RuleML tools can be adapted to SWRL. Currently the SWRL 2 specification (OĆonnor et al., 2012) is under development, to include OWL 2 features into SWRL.

W3C Rule Interchange Format (RIF) recommendation (Kifer, 2008; rif, 2008) is an alternative to RuleML, focused on rule interchanger. RIF was created for exchanging rules among rule systems, in particular among Web rule engines. It is an XML language for expressing rules which computers can execute. RIF was born with the philosophy that a single language would not satisfy the needs of many popular paradigms for using rules in knowledge representation and business modelling. Because of the serious tradeoffs in the design of rule language, RIF provides multiple versions, called dialects. RuleML and RIF recommendations influence each other, both providing input to the counterpart. Shared RIF RuleML implementations and use cases are projected to lead to further convergence. When compared to SWRL, RIF covers most SWRL features are in the RIF-BLD with the exception of "different-from", thus it should be possible to exchange most SWRL rules via RIF. RIF-BLD supports multiple-arity predicates, and SWRL is limited to unary and binary predicates. RIF-BLD has functions, SWRL is function-free. RIF-BLD has an extensive set of datatypes and built-ins, SWRL supports most of the same datatypes but has no built-ins.

SPARQL Inferencing Notation (SPIN) (spi, 2011) is yet another approach to bring a rule format to the Semantic Web. It is based on SPARQL Protocol and RDF Query Language (SPARQL) queries that are wrapped in an RDF format. So, SPIN rules can be executed (because they are rules) and stored within the semantic data, in the end point, since they are RDF. SPIN rules provide explicit support for constraint checking in addition to the native OWL inferencing mechanisms (that are enhanced too). Thus, they are commonly used for *Data Quality Management* (Fürber and Hepp, 2010) and *Data Reasoning and Inferencing* (Spohr et al., 2012; Pignotti and Edwards, 2012). SPIN is more expressive than SWRL because SPARQL is, extensible –functions and templates can be created– and *object-oriented.* Any SWRL rule can easily be converted to a SPARQL query, as an example, Listing 3.1 shown the implementation of the same rule in SWRL and SPARQL (SPIN).

Listing 3.1: Comparison between SWRL and SPARQL.

```
// SWRL rule
parent(?x,?y) \wedge brother(?y,?z) \Longrightarrow uncle(?x,?z)

// SPARQL rule
CONSTRUCT {?x :uncle ?z}
WHERE
{  ?x :parent ?y .
   ?y :brother ?z .
}
```

In addition, SPIN inherits the support and evolution of SPARQL –since it acts as an envelope for SPARQL queries. For all these reasons, some semantic suits dropped SWRL compatibility in favour of SPIN (Topbraid).

Notation3 (N3) is another alternative for describing rules in the semantic Web. N3 Logic (Berners-Lee, 2011) provides additional RDF properties that allow N3 to be used to express rules in a web environment. They were designed to be informal semantics that should be understandable by a human being. These properties are not part of the N3 language, but are properties which allow N3 to be used to express rules. Just as OWL is expressed in RDF by defining properties, so rules, queries, differences, and so on can be expressed in RDF with the N3 extension to formulae. However, N3 logic is still at a draft state, and it is not fully supported by semantic engines. The EYE reasoner (Meester et al., 2015) is a reasoning engine that uses an optimized resolution principle, supporting forward and backward reasoning. Backwards reasoning with new variables in the head of a rule and list predicates are a useful plus when dealing with OWL ontologies. EYE supports all N3's expressibility whilst being more performant than other N3 reasoner (Verborgh and De Roo, 2015).

3.3 Evented WEb ontology (EWE) model

In this section, we present the EWE Ontology, which models the most important aspects of TAS, and stands as a reference model to define and describe TASs. EWE is available online (Coronado and Iglesias, 2015b).

After analysing the TASs in the former chapter, we derive a model that contains the most relevant concepts (and relations between them) presented by those services, using the most widespread terminology. The EWE Ontology comprises two major parts: the main objects and properties defined in the EWE ontology, and the mapping that relates EWE to existing ontologies such as TAGS or FOAF(Amini et al., 2014). The ontology design methodology and examples of rules defined with EWE are also given in this section.

3.3.1 EWE design methodology

The final model we propose is the result of an iterative development process consisting of three steps: i) we analyse each of the TASs considered, identifying features and functionalities, then we extract the concepts and properties they address; ii) we define a model that

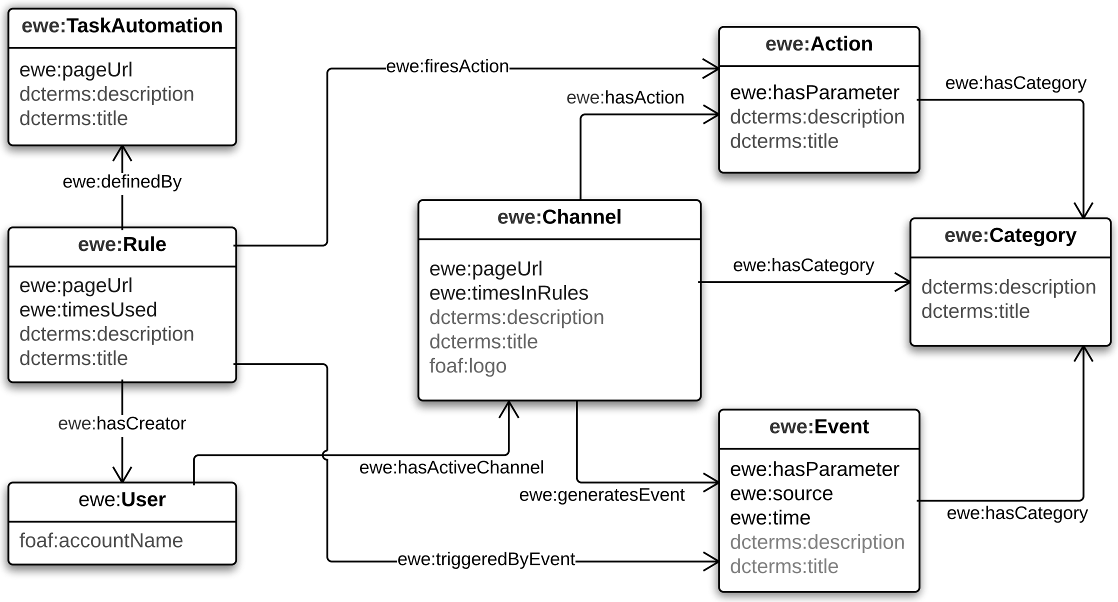


Figure 3.1: Detail of the EWE Ontology Model.

formally describes those elements; and finally, iii) we evaluate the model against the different use cases considered, i.e. we check how suitable it is for describing rules from sites such as Ifttt or Zapier. After each iteration, we repeat the process, including some new elements that the results have shown to be relevant, and remodelling others to best describe the domain. Each iteration refines the model, i.e. classes and properties are included, modified, or even removed from the ontology to make it not only broader but also more accurate, thus more useful.

Therefore, the ontology design has been undertaken by an iterative incremental development as in most agile software development methodologies. Figure 3.1 presents a diagram of an excerpt of the resulting model showing the major classes and properties.

3.3.2 EWE elements: main classes and properties

This section addresses the main objects and properties –which are the outcome of the design methodology– included in the EWE ontology.

3.3.2.1 Main classes

The core of the ontology comprises four major classes: Channel, Event, Action and Rule. The description of particular TASs or use case scenarios may inherit from them, creating new classes that are specific to the domain. We detail the usage of the main classes below.

The class *Channel* defines individuals that either generate Events, provide Actions, or both. In the context we refer to, Channel usually defines Web services. For instance, according to this definition *Dropbox* is channel because it generates Events every time a new file is uploaded or changed, and it provides the capability (Action) to rename files or move them to another folder. Of course, Dropbox channel would offer much more Events and Actions. Furthermore, sensors and actuators are also modelled as Channels since they produce Events and/or provide Actions; e.g. a wearable device with GPS programmed to generate alerts when it is near certain locations.

The class *Event* defines a particular occurrence of a process, which may trigger rules in the TAS. As opposed to the definition given in other ontologies (Raimond and Abdallah, 2012), in EWE events are instantaneous, i.e. they have no duration over time. The Event class may be subclassed to define particular types of Events. For instance, the class *NewChatMessage* is subclass of Event and defines the type of event that is generated when a new chat messaged is typed. Instances of Event class offer information of the particular event; e.g. instances of *NewChatMessage* have information of the chat message and the date when it was sent. Event individuals are generated by a certain Channel, when triggered by the occurrence of the process that defines them, e.g. typing and sending a message in *GoogleHangouts* triggers the generation of an event instance of type *NewChatMessage*. We say that the channel that generates the event is the event generator; *GoogleHangouts* is the event generator. Definitions of Event subclasses are not bound to a Channel since different channels may generate the same events; e.g. both, *GoogleHangouts* and *Whatsapp* channels may generate instances of *NewChatMessage*.

The class *Action* defines an operation or process provided by a Channel. Actions produce effects whose nature depends on the action nature. These include: producing a log message, modifying a public or private state on a server, a physical action such as switching on a light, etcetera. The effect can even trigger an Event generation, either by the same Channel or a different one. Similar to Event, the Action class may be subclassed to specific actions. Again, Action definitions are not bound to a Channel, because different channels can support the same actions; e.g. *Linkedin* and *Facebook* channels provide the *ChangeProfilePicture* action.

The class *Rule* defines an automation, i.e. an ECA rule triggered by an Event that produces the execution of an Action. Rules define particular interconnections between instances of the Event and Action classes transferring information from the event to the action. The aim of EWE's Rule is describing automations, giving information that may be used to compare automations, to analyse datasets of automations, etcetera. Thus, Rule class does not provide explicit support to rule execution, it delegates this to any of the recommendations described in Section 3.2.3, e.g., SPIN, N3 Logic or RIF.

3.3.2.2 Main Properties

In addition to the description of the main classes, we offer a list of the important properties and the purpose of including them within the overall EWE ontology. These are *hasParameter*, *hasCategory*, *activeChannel*, *hasCreator*, *firesAction* and *triggeredByEvent*.

The property *hasParameter* presents the parameters of an Action or an Event. Instances of Event and Action can rarely be defined without some configuration parameters. They differentiate each individual, e.g. the body and the sender of an *EmailReceivedEvent*. Each parameter should be provided by the appropriate *subPropertyOf* of *hasParameter*. Given the *hasParameter* example, the body is given by a *hasBody* property, and the sender by a *hasSender* property both sub properties of hasParameter.

The property *hasCategory* indicates that a Channel, Event or Action belongs to a certain category. An element may have more than one category. The EWE Ontology does not provide a taxonomy of channels, events, or actions; but it facilitates building that classification. The Channel categorisation is important for channel discovery and recommendation. This happens not only with discovery and recommendation methods based on profiling, but also with methods based on semantic similarity. Besides, expert systems may use the channel categorisation and help a user to find alternatives to other channels; e.g. when a channel is not available due to geographical restrictions. The range of the hasCategory property is the class Category, a subclass of the *Concept* class from the SKOS(Amini et al., 2015) ontology.

The *hasActiveChannel* property links users to Channel on which they have an account, i.e. a channel they can include in the Rule definition. Combined with the category of the channels, this information may be used to reason over the alternatives that a user may have to a particular choice of channel.

The property *hasCreator* links instances of Rule to its creator, an *onlineAccount* from the FOAF ontology. The authors of the rules are significant information for data analysis purposes and recommendation systems.

The *ewe:firesAction* and *ewe:triggeredByEvent* properties link Rule instances with the event and action instances that are related to it. They describe the rules enabling rule comparison and analysis.

3.3.2.3 Channel characteristic dimensions properties

Apart from the features extracted using the mentioned methodology, we included several additional properties to represent the characteristic dimensions of channels, rules and execution profiles studied in the former chapter.

The property *channelType* classifies the channel. Its range is the class *channelType*, and according to the classification of channels made in the former chapter. There are two individuals of class channelType: *webChannel* and *deviceChannel*.

Regarding the privacy paradigm, the property *privacyType* defines the privacy policy associated to the channel which, as explained, defines the scope of the events and actions that channel produces. With range *channelPrivacy*, three individuals are defined: *publicChannel*, *privateChannel*, and *privateGroupChannel*.

Regarding the configuration paradigm, the property *hasConfiguration* defines which parameters should be provided by the user to activate the channel. Its range *channelConfiguration*, with the properties dc:title and dc:type, and dc:description.

The property related to the configuration paradigm has already been described, the hasParameter property, because it also appeared as a feature given by the ontology design methodology. In this case, the domain is the union of the Event and Action classes.

There are two properties related to the communication paradigm of device channels: communicationType and coverageRestriction. The property *communicationType* indicates how is the communication with between the devices carried out, and the property *coverageRestriction* –is present– sets the restrictions of coverage of channel activity. In both cases, the range of the property is not defined because these are declarative properties to describe the communication type and coverage. This ontology does not intend to create a categorisation of coverage types o communication types. Those grounding features, such as the communication protocol or frequency band, should be described by a different grounding oriented ontology.

Regarding the discovery paradigm, the *isDiscoverable* property is included to indicate whether the channel include any mechanism that allows the TAS to discover it –thus, it is a boolean datatype property. The description of the discovery methods should be modelled with a different ontology.

Finally, the execution profile of the TAS is given by the *hasExecutionProfile* property, whose domain is the TaskAutomationService

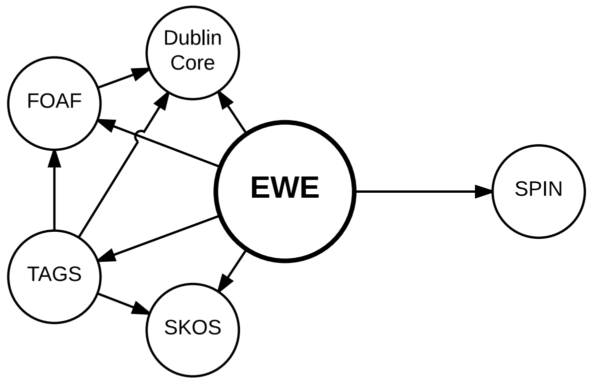


Figure 3.2: External ontologies mapping.

3.3.3 Mappings from external ontologies in EWE

EWE has been developed based on a number of existing classes and properties from external ontologies as possible in the spirit of the linked data philosophy. Hence, EWE enhances search, interoperability and linking data because several tools and search engines are capable of using those vocabularies. The connections between EWE and external ontologies are summarised in Figure 3.2 and detailed below.

- **SPIN**: the contribution of SPIN to EWE is essential, since rule grounding is done by means of *sp:Construct* instances. The implementation of *SPIN* rules as members of EWE rules connects Events to Actions in the same way they are connected in TASs.
- **FOAF**: EWE *User* class inherits from FOAF *onlineAccount* and is connected by an *account* to a FOAF *Agent*. This assures better interoperability and search operation, as well as enhances profiling by allowing using external context.
- **TAGS**: the TAGS ontology provides a common definition of tags. This ontology can extract information from third party sources and also can provide smart connection and recommendations depending on the tags set. The *taggedWithTag* property, from the TAGS ontology, links rules to each of their associated tags. This is thus another method for classifying Rules.
- **SKOS**: SKOS is proposed for representing structured vocabularies for the Semantic Web. Several tools search and reason using the relations they define. The EWE concept *Category* inherits from the SKOS Concept.
- **dcterms**: most of the administrative properties of every class of EWE are defined according to DCMI Metadata Terms (dcterms) metadata elements. This improves the

searches of elements of the ontology.

3.3.4 Examples of EWE use

In order to give a better idea of how specific Channels, Events and Actions are defined using EWE; we show a short excerpt of a Gmail channel definition in Listing 3.2. The channel represents the Gmail channel implemented in a particular TAS (e.g. Gmail channel from Ifttt[1]). We use the punning mechanism of OWL2 (Ruttenberg et al., 2008) to attach properties to that channel. As a result, the description explicitly states that the channel may generate events of a particular class (*ewe-mail:NewEmail*) and provides a particular action for them (*ewe-mail:SendEmail*). This is useful to analyse and compare data from different TASs. The example also includes a few related *hasParameter* subproperties.

Listing 3.2: Channel definition excerpt.

```
ewe-mail:Gmail a owl:Class ;
    rdfs:subClassOf     ewe:Channel ;
    rdfs:type           ewe:Channel ;
    dcterms:title       "Gmail"^^xsd:string ;
    dcterms:description "Webmail by Google"^^xsd:string ;
    ewe:generatesEvent  ewe-mail:NewEmail ;
    ewe:hasAction       ewe-mail:SendEmail .

ewe-mail:NewEmail a owl:Class ;
    rdfs:subClassOf     ewe:Event ;
    dcterms:title       "Any new email in inbox"^^xsd:string ;
    dcterms:description "New email arrives in Gmail"^^xsd:string .

ewe-mail:SendEmail a owl:Class ;
    rdfs:subClassOf     ewe:Action ;
    dcterms:title       "Send an email"^^xsd:string ;
    dcterms:description "Send an email from Gmail"^^xsd:string .

ewe-props:hasBody       rdfs:subPropertyOf :hasParameter.
ewe-props:hasSender     rdfs:subPropertyOf :hasParameter.
ewe-props:hasToAddress  rdfs:subPropertyOf :hasParameter.
ewe-props:hasSubject    rdfs:subPropertyOf :hasParameter.
```

An example of event and action instances with grounded parameters, which are based on the concepts defined in the listing given above, is presented in Listing 3.3.

[1]Ifttt Gmail channel website: http://ifttt.com/gmail.

Listing 3.3: Event and action instances.

```
:new-email#1 a ewe-mail:NewEmail ;
    ewe-props:hasSubject    "the email subject"^^xsd:string ;
    ewe-props:hasBody       "the email body"^^xsd:string ;
    ewe-props:hasSender     "from-this@email.com"^^xsd:string .

:send-email#1 a ewe-mail:SendEmail ;
    ewe-props:hasSubject    "the email subject"^^xsd:string ;
    ewe-props:hasBody       "the email body"^^xsd:string ;
    ewe-props:hasToAddres   "to-this@email.com"^^xsd:string .
```

Similarly, automation rules are described using the punning mechanism to attach classes to properties of Rule instances. In the example shown in Listing 3.4, the rule instance describes a rule that is triggered by events from class *NewEmail* and produces actions of class *SendChatMessage*, in that example the implementation of the rule is given using the *spin:rule* property. However, this is not part of the EWE ontology, and so other rule description vocabularies may have been used.

Listing 3.4: Rule instance.

```
:email-alarm#1 a ewe:Rule ;
    dcterms:title    "Text me when I receive an important email"^xsd:string ;
    ewe:triggeredByEvent    ewe-mail:NewEmail ;
    ewe:firesAction         ewe-chat:SendChatMessage ;
    spin:rule               :rule#1 .
```

As explained, rules are conveniently defined as SPARQL construct queries, and then stored as SPIN objects in RDF format. Since SPIN syntax is more complex than SPARQL syntax, for the sake of readability, we present the implementation of the rule from Listing 3.4 in SPARQL format, shown in Listing 3.5. The example shows a rule that is triggered every time an event of type *NewEmail* happens. Events are filtered so only those with the label 'important' are considered. Then, a *SendChatMessage* action is generated. The message sent contains the email address captured from the incoming event. As seen, the event-condition part of the ECA rule is given by the WHERE clause and the FILTER function. The action part is given by the CONSTRUCT clause, using the variables bound in the WHERE clause.

Listing 3.5: Rule in SPARQL form.

```
CONSTRUCT {
  ?action a ewe-chat:SendChatMessage .
  ?action ewe-props:message ?msg .
}
```

```
WHERE {
  ?event a ewe-mail:NewEmail .
  ?event ewe-mail:FromAddress ?fromAd .
  BIND (fn:concat("Important message from: ", ?fromAd) AS ?msg) .
  FILTER {
    ?event ewe-props:haslabel "important"
  }
}
```

The rule of the example does not restrict the incoming event to be of a particular Channel. It will be triggered by events of the appropriate type regardless of what channels generates them: the Gmail channel, the Yahoo channel, or any other. Therefore, EWE allows defining rules where the Channels, Events, and Actions involved are not bound. This example illustrates how EWE enables a new kind of rule, looser than the classical rules defined by the TASs explored in the related works section.

3.4 A prototype of a semantic TAS with EWE

This section illustrates the use of EWE to implement a semantic TAS which enables reasoning over LOD. For that purpose, we have developed a prototype of a TAS with several channels integrated within the platform. These channels generate events that are described following the EWE model and processed by the engines using SPIN rules. This section also contributes to give the reader an explanation of i) how events are distributed in a semantic TAS based on EWE, ii) how Actuators and Web Services handle actions, and iii) how the Rule Engine processes incoming events and fires rules. Nevertheless, the prototype described in this section is shown as a use case, and it is not intended to be an implementation of the reference architecture shown in the former chapter, nor to achieve great results in terms of performance, scalability, etcetera.

3.4.1 Prototype architecture and operation

Figure 3.3 presents the general architecture of the prototype whose components are distributed in three layers. The generation layer shows that events may come from either Web Services or Sensors, thus there are channels of different nature. Since several processing modules take part in the execution flow, and in order to support other modules to be included in the future, a transportation layer is needed to distribute these events among them. Engines in the processing layer are in charge of executing the rules when their triggers ap-

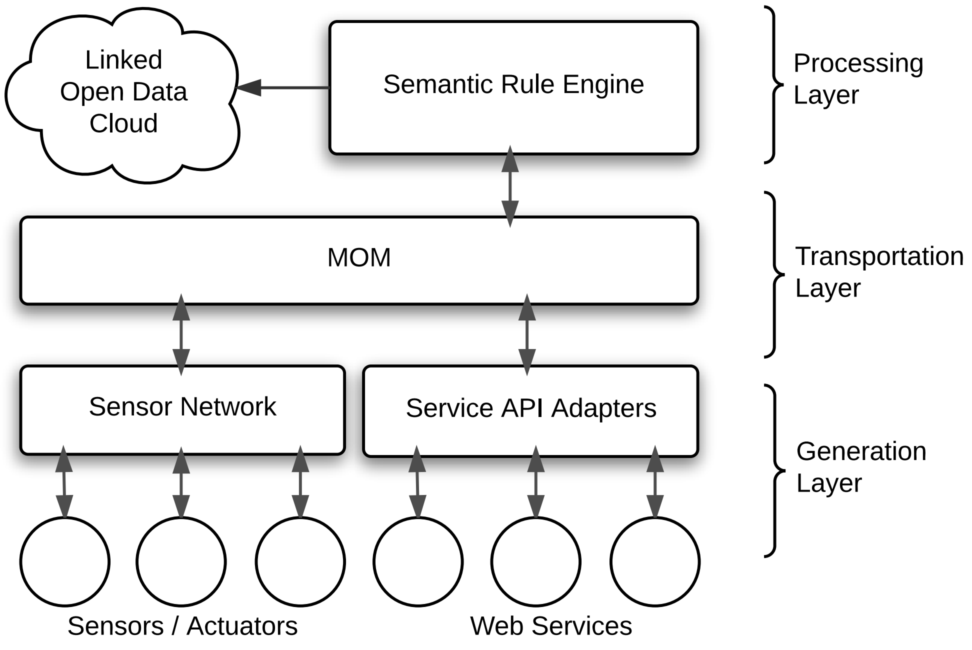


Figure 3.3: General architecture of TAS implementation with EWE support.

pear. In particular, the Semantic Rule Engine is able to connect to SPARQL endpoint of LOD cloud and to use that information to evaluate the rule conditions.

When Web Services and Sensors generate events, the Service API Adapters or the Sensor Network, respectively, generate a message that is pushed to the Message Oriented Middleware (MOM) in the transportation layer. A payload with the RDF representation of the event has to be included in these messages. The MOM is in charge of distributing the messages to the subscribers according the publish-subscribe pattern (Eugster et al., 2003). Thus, the Semantic Rule Engine in the processing layer subscribes to the event messages from the MOM. In the same manner, the Sensor Network and the Service API Adapters subscribe to the action messages. When these two modules receive the action messages from the MOM, they forward these messages to the Actuators or the Web Services, respectively, which are responsible for interpreting and executing the actions. This communication based on messages gives loose coupling to the architecture and fits the Semantic Web vision (Berners-Lee et al., 2001).

The Semantic Rule Engine is responsible for: processing the incoming event messages; executing the rules; and discarding the messages once they have been processed to avoid executing the same rule multiple times for the same event. In our prototype, which employs the SPIN notation and a SPARQL engine, the manner that rules are processed relies on the fact that these rules can be expressed as SPARQL queries (see example in Listing 3.5).

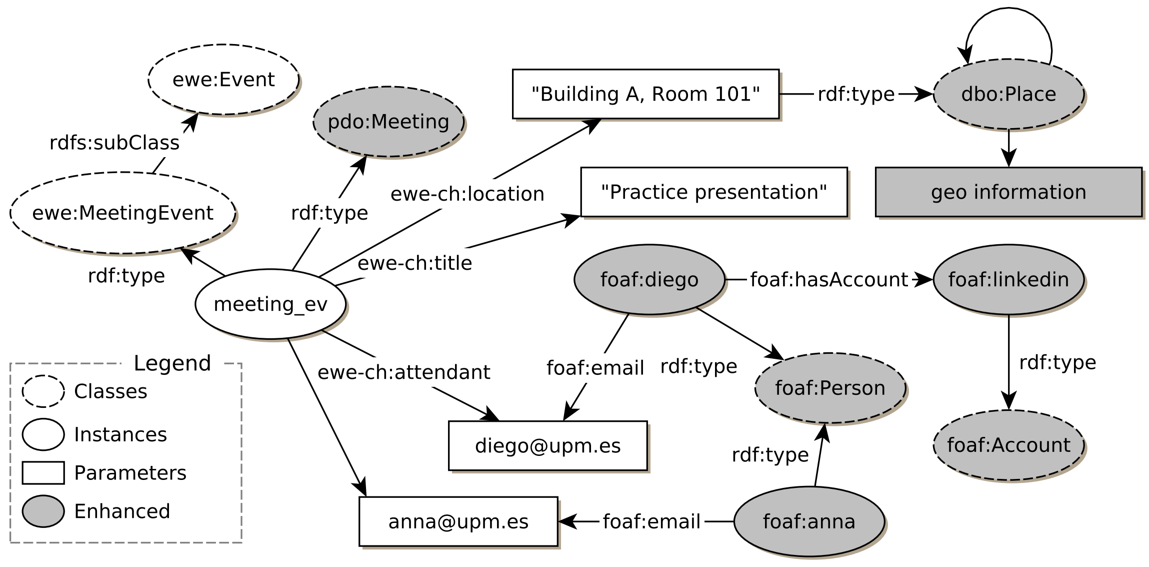


Figure 3.4: Use case example event model with linked-data.

These SPARQL queries are run every time an event is received in the SPARQL engine. After executing the rules in their SPARQL query form, the new actions constructed by these queries are pushed to the MOM. Then the MOM can deliver these actions to the subscribers, e.g. Sensor Networks and Service API Adapters.

In order to provide a complete overview of how an event is generated, processed by the rule engine, and the constructed action is executed; we illustrate all the steps for processing the rule "Whenever I receive an email labelled as important, text me a notice" shown in Listing 3.5. First of all, a user (e.g. Ann) should have created this rule. Then, the rule is loaded in the Semantic Rule Engine. When Ann receives an email in her Gmail account, the Service API Adapter, which is connected to the Gmail service, generates an *ewe-mail:NewEmail* event message that is pushed to the MOM. Then, the MOM distributes the new message to the subscribers. In our case, the Semantic Rule Engine receives the message, extracts the RDF payload, and executes the rule. In case the email has the label 'important', a new action message will be constructed (*ewe:NewChatMessage* action) and pushed to the MOM. Consequently, the MOM distributes the new action message to the subscribers. In the example, the Service API Adapter receives the message, extracts the service parameters, and sends a request to the Hangout Service API, which texts Ann informing about the important email just received.

Event generator (channel)	Enhancements	Vocabularies
Blogger, ESPN, Evernote	Post NLU. Emotion analysis	sioc, onyx, sport
Delicious, Evernote	Tagging and categorisation	tags, skos
Facebook, Linkedin, G+	Unified account information	foaf, vcard, curric
Foursquare, Life360	Geolocation information	geo
Last.fm, SoundCloud	Track recognition. Music emotions	music, af, onyx
Wemo, SmartThings	Sensor information, deployment loc.	ssn

Table 3.1: Examples of vocabularies that may be used to enhance events.

3.4.2 Semantic use case scenario

Based on this general architecture for a prototype, we have modelled a scenario about meeting arrangement that takes advantage of the semantic capabilities of EWE. Figure 3.4 shows an instance of a meeting event similar to those the semantic engine work with. As seen in the figure, some of the classes and instances are labelled as *Enhanced* (according to the legend). We refer to types, properties and instances that are not directly provided by the event generator (the Web Services or Sensors), but obtained from semantic endpoints (from the LOD cloud). The information retrieved is sometimes several steps away from that information initially given. For instance, with the email address of the attendees, we can derive: their social ids described with the FOAF ontology; then their Linkedin account; and, then their public profile. Similarly, the location of the meeting, its agenda, or the project to which the meeting is related; may be retrieved from the semantic endpoints.

There are numerous vocabularies available that we can use to describe different types of events as well as related information that may be derived from the data LOD cloud. In Table 3.1, we present several channels that are available in Ifttt. For each of them, we suggest how the information provided by EWE may enhance the TAS as well as which mappings could be used.

This enables the definition of rules that include clauses in the condition part that are not directly related to the information given within the event. We could define a rule that reads "When I confirm the attendance to a meeting, add all other attendees to linkedin as contacts". In this case, the Semantic Engine would have to browse the LOD to get the foaf:Agent of the attendees, and in case they have a foaf:OnlineAccount that describes a Linkedin account, obtain the information needed to add them as contacts. This is an enlightening example of reasoning over large scale data outside the platform which is possible thanks to the contribution presented in this chapter, the *Evented WEb ontology* (EWE) model. The implementation of the semantic TAS used in this section is available online (Crespo et al., 2014).

3.5 EWE evaluation

In this section, we describe the evaluation process that was carried out to show how EWE success at modelling some of the TASs revised in the related works section while the shortcomings of these approaches are addressed.

According to Brank et al. (Brank et al., 2005), when evaluating an ontology is more practical to focus the evaluation of different levels of it separately. They identify several levels —lexical, hierarchical, semantic, application, etcetera— and determine which evaluation approaches are more suitable for evaluating each level. In general, selecting which levels should be evaluated depends on the nature of the ontology itself. In our case, the EWE ontology was designed to describe all relevant features from existing TASs. Altogether, those TASs form a wide domain to be modelled, which possesses important differences between their feature sets. Hence, the evaluation described will focus on the *lexical* level. Besides, in Section 3.4 we show an architecture based on EWE to improve the capabilities of a typical task automation scenario. Thus, that use case addresses the *application* level evaluation.

Since there is no golden standard to compare with, a data-driven evaluation was selected. To do so, we first analysed a set of selected TASs and extracted their main features to define a corpus; then, we define and apply metrics to get the results from the corpus.

3.5.1 Feature extraction

The feature extraction process consists of the formalisation of a conceptual model as a list of features. Then, we will use those extracted features to compile the corpus to use. In the process, we thoroughly analysed Ifttt, Zapier, on{x}, and Tasker, to obtain four theoretical models that represent them. Ifttt and Zapier were included because they have many features in common with most of the TASs we analysed in Section 2.4, but Tasker and on{x} were also considered because they have many unique features.

We capture the feature list of each TAS by applying Algorithm 1 to the corresponding

Algorithm 1 Feature extraction process.

```
Data: Conceptual model of the Task Automation Service
Result: Formal list of features related to the Task Automation Service
foreach Concept in Model do
    addFeature: $\exists$ Concept
    foreach Property of Concept do
        addFeature: hasProperty(Concept, Property)
    end
    foreach Relationship of Concept do
        addFeature: hasRelation(Concept, Relationship)
        foreach Restriction of Relationship do
            addFeature: restriction(Concept, Relationship, Restriction)
        end
    end
end
```

conceptual model. As it shows, we considered features of four types: *concepts*, *properties*, *relationships* and their *restrictions*. A sample of the feature extraction process outcome is shown in Table 3.2. The summary of the results of the process carried out with the four TASs selected is also presented in Table 3.3.

With the whole list of features extracted, we compile the corpus to use in the evaluation. The list of features extracted from each TAS is modelled as a logical vector where each position represents a feature from the corpus: '1' or '0' on a position means that the feature is in the TAS or is not included, respectively.

$$\mathcal{T}_j = (f_1, f_2, ..., f_n), f_n \in \{0, 1\} \tag{3.1}$$

The EWE ontology is also represented as a logical vector in order to apply the metrics as explained below.

3.5.2 Running the evaluation

We use the *coverage* and *accuracy* functions to the measure of the quality of the ontology. Coverage measures the spread of the ontology: the more features included, the greater the coverage. We also use the accuracy function, to keep the ontology as sharp and concise as possible. Both of these are in a trade-off between embracing many features, and perfectly fitting the corpus.

Feature	Type
∃ Rule	Concept
hasProperty(Rule, title)	Property
hasProperty(Rule, dateCreation)	Property
hasProperty(Rule, timesUsed)	Property
hasRelation(Rule, tagged)	Relation
restriction(Rule, tagged, range(Tag))	Restriction
hasRelation(Rule, hasCreator)	Relation
restriction(Rule, hasCreator, range(User))	Restriction
hasRelation(Rule, triggeredBy)	Relation
restriction(Rule, triggeredBy, range(Event))	Restriction
restriction(Rule, triggeredBy, maxCardinality(1))	Restriction

Table 3.2: List of features extracted from Ifttt rules model.

Feature type	Ifttt	Zapier	Tasker	On{x}
Concepts	11	14	16	21
Relations	12	24	18	26
Properties	22	13	30	36
Restrictions	31	37	32	40
Total	76	88	96	123

Table 3.3: Feature extraction process outcome.

Target	Coverage	Accuracy
Ifttt	0.96	0.91
Zapier	0.92	0.84
on{x}	0.42	0.38
Tasker	0.51	0.48

Table 3.4: Coverage and accuracy results for the EWE ontology.

Coverage describes the range of relevant features from the domain in use that are defined in the ontology. Currently, the state of the art has different ways for evaluating the coverage of an ontology over a knowledge domain. The straightforward approach is measuring the size of the ontology, i.e. counting classes. This can be sufficient for a quantitative view (Ruan and Yang, 2010), even when it lacks the qualitative aspect (Ouyang et al., 2011).

Therefore, coverage is defined as the proportion of features from the TAS that are defined in the ontology. With $\mathcal{T}$ being the TAS and $\mathcal{O}$ the ontology:

$$Coverage(\mathcal{O},\mathcal{T}) = \frac{Dim(\mathcal{O} \cap \mathcal{T})}{Dim(\mathcal{T})} \tag{3.2}$$

Accuracy is defined as the Jaccard similarity (Jaccard, 1901).

$$Accuracy(\mathcal{O},\mathcal{T}) = \frac{Dim(\mathcal{O} \cap \mathcal{T})}{Dim(\mathcal{O} \cup \mathcal{T})} \tag{3.3}$$

We introduced the accuracy metric to keep the ontology as tightly suited to the corpus as possible, i.e. as simple and useful as possible. If the ontology embraces too many features, it becomes huge, too complex, and thus hard to use.

The results of the process are generated using formula (3.2) and formula (3.3), and are summarised in Table 3.4. As anticipated, the results are favourable to Ifttt and Zapier, which shows our modelling efforts to focus on the common TASs features. Including most of the features from Tasker or on{x} in EWE would have increased their coverage measure at a cost of lowering the accuracy metric of Ifttt and Zapier.

Then, we undertook a scrapping process to extract data of channels, events, actions and rules from the websites of Ifttt, Zapier, on{x} and Tasker. In this case, the data is extracted from the application itself. The results showed 55,914 rules connecting 718 events to 910

Class / Individuals	Ifttt	Zapier	on{x}	Tasker
Rule	80353	31501	39	-
Channel	195	521	-	22
Event	725	1532	70	72
Action	357	1046	352	218

Table 3.5: Scraping process results.

actions that were provided by 222 different channels. To extract the data from the websites, we used *Scrappy* (Fernández-Villamor et al., 2012), since it returns data in RDF and can be instructed to format it according to EWE.

The crawling, made during the last quarter of 2015, obtained the results that are summarised in Table 3.5. Although the four TASs studied are similar, the results show great differences in the ratios between channels, events, and actions. Ifttt and Zapier, both web platforms, present around two actions per channel on average and a higher ratio of events. Tasker and on{x}, both smartphone apps, show opposite results. They present a greater number of actions than events: three times the number of events for Tasker, and five times for on{x}. This is because their control over the device allows them to provide many actions.

Finally, the scraped data were loaded into a SPARQL endpoint in order to make them available as a directory of channels, events and actions available in several TASs. We defined a set of meaningful queries that a common user would make in order to retrieve information about the channels available, the events of a particular type, etcetera. We coded those queries using SPARQL and executed them. Table 3.6 gathers some of them together with their execution outcomes.

These experiments show that the EWE ontology this chapter contributes with effectively models the TAS domain; i.e. the classes and properties defined by EWE are suitable for describing TAS channels, actions, events and rules. This conclusion is not only supported by the metric results shown in Table 3.4, but also by the process of automatically extracting data from TASs websites and applications. This process is described in detail in Section 4.3.1 in Chapter 4 [2]. Finally, the scenario described around the architecture and prototype proposed in Section 3.4 supports the hypothesis that semantic TASs address the identified

[2]Author's note: the instance extraction is an essential part of ontology learning process, so we described it there for convenience

Query	SPARQL Query	Results
How many channels are supported by each TAS?	`SELECT ?tas (COUNT(?serv) AS ?channels) WHERE { ?serv rdfs:subClassOf ewe:Channel . ?serv ewe:supportedBy ?tas } GROUP BY ?tas`	Zapier (141), Tasker (22), Ifttt (65)
Which channel categories are defined?	`SELECT DISTINCT ?category WHERE { ?serv rdfs:subClassOf ewe:Channel . ?serv ewe:hasCategory ?category }`	Event Management, CRM, Phone, Developer Tools, Email Marketing, Social and 9 more
Which channels are categorised as social?	`SELECT ?servName WHERE { ?serv rdfs:subClassOf ewe:Channel . ?serv ewe:hasCategory ewe:Social . ?serv dcterms:title ?servName }`	Wordpress, buffer, chatter, Facebook, Twitter, and 7 more
How many actions can be executed by means of the Tasker TAS?	`SELECT (COUNT(?act) AS ?actsProvided) WHERE { ?chan ewe:hasAction ?act . ?chan ewe:supportedBy ewe:Tasker }`	204 actions
How many rules are triggered by Gmail channel events?	`SELECT (COUNT(?rule) AS ?ruleCount) WHERE { ?srule rdf:type ewe:Rule . ?srule ewe:triggeredByEvent ?Event . ewe:Gmail ewe:generatesEvent ?Event . } GROUP BY ?rule`	7240 rules

Table 3.6: SPARQL example queries using the EWE ontology.

TASs drawbacks.

3.6 Discussion and Outlook

The EWE ontology presented in this chapter constitutes a specification of a reference model for describing TAS. As a semantic model, it presents several benefits ranging from providing better communication of concepts around TASs to enabling reasoning over LOD capabilities for the automation rules. It establishes the foundations of building a semantic TAS.

This chapter central contribution is the definition of the *Evented WEb Ontology* (EWE). EWE constitutes a specification of a reference model for describing TAS. As a semantic model, it presents several benefits ranging from providing better communication of concepts around TASs to enabling reasoning over LOD capabilities for the automation rules. It establishes the foundations of a semantic TAS. The chapter introduces a realistic use case scenario where the progress made beyond the state of the art is illustrated. Besides, it presents an implementation of a semantic TAS with support to EWE rules. This prototype employs *SPARQL Inferencing Notation* (SPIN) to describe *Event-Condition-Action* (ECA) rules. Finally, the chapter exhaustively evaluates the EWE ontology proposed by using a data-driven approach. The evaluation conclusion is that EWE effectively models four popular commercial TASs while it addresses the shortcomings observed in them.

For sake of reproducibility, the material developed within this work, including code, is available online for the interested reader: the EWE ontology (Coronado and Iglesias, 2015b), and the implementation of the semantic TAS (Crespo et al., 2014).

Our main future work is to develop an agent system to assist users to configure new rules in the semantic TAS proposed. To feed the agent system knowledge and thus to enhance the assistance provided, a clustering method can be proposed to group channels, events, actions and rules. This agent system may recommend related automation rules and variations given a certain rule. Given a channel, it will bring up those similar channels according to different criteria: frequently connected in rules, same category, similar event and action sets, etcetera. This clustering approach can be used to infer Channel groups, as given by some TASs. We aim to create a disambiguation method that combines channels which are equal but described in different TASs. This method may determine which events and actions correspond to the same purpose to merge them. The EWE ontology has to be adapted to support this issue. Besides, some TASs providers have shown their interest in building a tagging system for rules, as already covered by EWE. In this regard, we intend to extract and analyse the tags when available in order to incorporate them into the expert system knowledge.

Another important future work is the evaluation of alternative manners for passing messages and their performance in the proposed TAS architecture. In this vein, to further explore SPARQL Streaming technologies (Bolles et al., 2008; Barbieri et al., 2009) and the feasibility of processing RDF events in real time, we will extend the prototype presented in this chapter with the best alternative assessed. This is not required in a low rate event scenario, as the one described in Section 3.4. Nonetheless, real time events are desirable in case of having sensors or other high rate streams.

CHAPTER 4

Mining TAS's channels: An ontology discovery approach

This chapter presents a methodology for automatic ontology learning in the context of TAS. This methodology complements the EWE ontology, since it provides a mechanism to extract automatically instances of Channels and Automations from commercial TAS, and proposes an algorithm to generate vocabularies derived from these instances. Moreover, it reduces the maintenance cost of these vocabularies being an unsupervised approach.

4.1 Introduction

The EWE ontology presented in the former chapter constitutes a specification of a reference model for describing TAS. As a semantic model, it presents several benefits ranging from providing better communication of concepts around TASs to enabling reasoning over LOD capabilities for the automation rules.

One of the most immediate usages of EWE consists in describing all channels from commercial TASs using the ontology. Being a semantic dataset, it allows agents and users to make complex queries about the channels supported by each TAS, the event and actions provided by each channel, and also to reason over them. However, in spite of their benefits, semantic models have a high maintenance cost (Sabou et al., 2005), and given the growing rate of the channel directories of TASs[1], an automatic approach for extraction those vocabularies is required. Furthermore, to fully benefit from the capabilities of semantic TAS these vocabularies must include mappings to external vocabularies and link external resources. The linkage process is also time consuming and in many cases repetitive.

Automatic ontology learning stands as a solution to these challenges. Being able to generate vocabularies from a dataset of instances at any domain of knowledge, ontology learning reduces the maintenance cost, and can handle the process of linkage to external instances. The resulting vocabularies best fit the domain they model (Sabou et al., 2005), specially when the process is tailored to a particular domain. More generalistic approaches may be found in the state of the art to generate new ontologies for diverse domains (Sánchez, 2012; Wong et al., 2012; Buitelaar et al., 2005). But this is not the case of the process described in this chapter, which focuses on reducing the maintenance cost of existing ontologies of the TAS domain.

Therefore, this chapter proposes an ontology learning methodology in the TAS domain, for discovering vocabularies of specific channel domains. Its main contributions are i) proposing an automatic process for describing all channel information from a TASs using the EWE ontology, and creating different mappings for concepts extracted from channels to other ontology concepts; ii) providing a similarity metric based on the semantic structure of channels in combination with their linguistic similarity, and design an algorithm that computes that metric; and iii) developing an algorithm for automatic channel ontology learning from the extracted instances using the proposed similarity metric. As a result, a pack of channel ontologies were learned. With these contributions it will be feasible to port rules from a TAS

[1]Ifttt increased its number of supported channels from 97 to 238 between the first quarter of 2013 and the last quarter of 2015.

to another, to swap events or actions in automations of two compatible channels, and to facilitate channel discovery. Beside, they support cross platform user profiling based on the automations they use, which facilitates rule recommendations and assistance in automation orchestration.

The remainder of this chapter is structured as follows: first, in Section 4.2 we introduce the background on automatic ontology learning. Then, in Section 4.3 we describe the methodology for ontology learning that comprises the process of extracting instances, the metric definition used to compute channel similarity, and the algorithm for ontology learning and generation. Next, in Section 4.4 we describe the evaluation process carried out to assess the methodology outcome. Finally, the conclusions and future work are stated in Section 4.5.

4.2 Background in Ontology learning

Manual ontology development is time consuming and error prone, and requires an expert in the domain of application. Moreover, since they are designed for a specific application they are hard to reuse (Sabou et al., 2005; Shamsfard and Barforoush, 2003). To get over these disadvantages, automatic ontology generation and ontology learning are suitable mechanisms to build ontologies from datasets or other domain models without any or very little supervision. They reduce the time and effort invested in ontology development, but also produce models that best fits their application (Sabou et al., 2005).

Ontology learning refers to extracting conceptual knowledge (represented as ontological elements) from the input dataset or domain model, and building an ontology from them (Shamsfard and Barforoush, 2003). It serves the purpose of supporting an ontology engineer in the task of creating and maintaining an ontology (Cimiano et al., 2009). The process covers several steps ranging from domain terminology extraction and concept discovery to learning ontology relations and populating the ontology. Throughout this process several technologies take part, such as Natural Language Processing (NLP), Information retrieval, semantic web, machine learning, etc.

Ontology learning systems may be categorised according to type of data from which they learn: there are structured, semi-structured and unstructured approaches (Hazman et al., 2011; Cimiano et al., 2009). The central problem in learning from structured data is to determine which pieces of structural information can provide relevant knowledge. For instance, a database schema may be used to identify ontology concepts and their relationship (Kashyap, 1999; Drumond and Girardi, 2008)

Unstructured and semi-structured ontology learning require preprocessing the input to extract the concepts, and learn the relations and attributes. The unstructured techniques feed from plain text, and so it constitutes the most difficult approach. They usually involve heavy NLP, combined with statistical analysis (Sánchez and Moreno, 2004), pattern matching, entity discovery (Tiddi et al., 2012) or machine learning(Cimiano and Völker, 2005). The most common case of semi-structure ontology learning uses web pages as input. It uses both traditional data mining and web content mining techniques to preprocess the data. Some approaches simply use the web pages structure, while others apply NLP to refine the structure. Then, a clustering algorithm is applied to extract the concepts and its relations. On the other hand, highly structured data as found in databases facilitates the application of pure machine learning techniques (Cimiano et al., 2009), or a set of rules for converting entities, attributes and relations in the database into concepts, attributes and relations in the ontology (Jacinto and Antunes, 2012). Likewise, semantic models and linked open data may be used for learning new ontologies or refine existing ones (Tiddi et al., 2012).

4.3 Methodology

In this section we present the methodology we follow to learn vocabularies from the information provided in the list of channels of each TAS's website. The process comprises four sequential steps as shown in Figure 4.1: Semantic scraping, calculate channel similarities, compute channel clusters, and generate the vocabularies. This strategy is common in information retrieval from unstructured data (Liu et al., 2013), it consists in progressively enhance the information until the knowledge is obtained. Throughout the rest of the section, we will describe each step in detail.

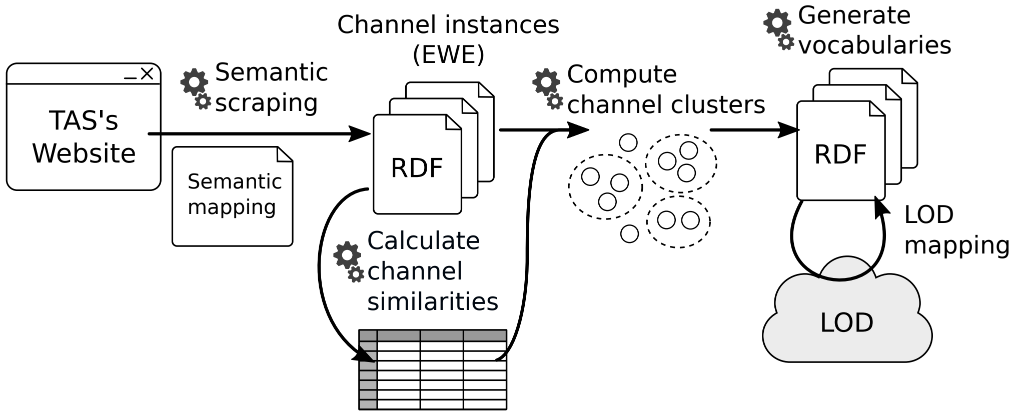


Figure 4.1: Overview of the ontology learning methodology used.

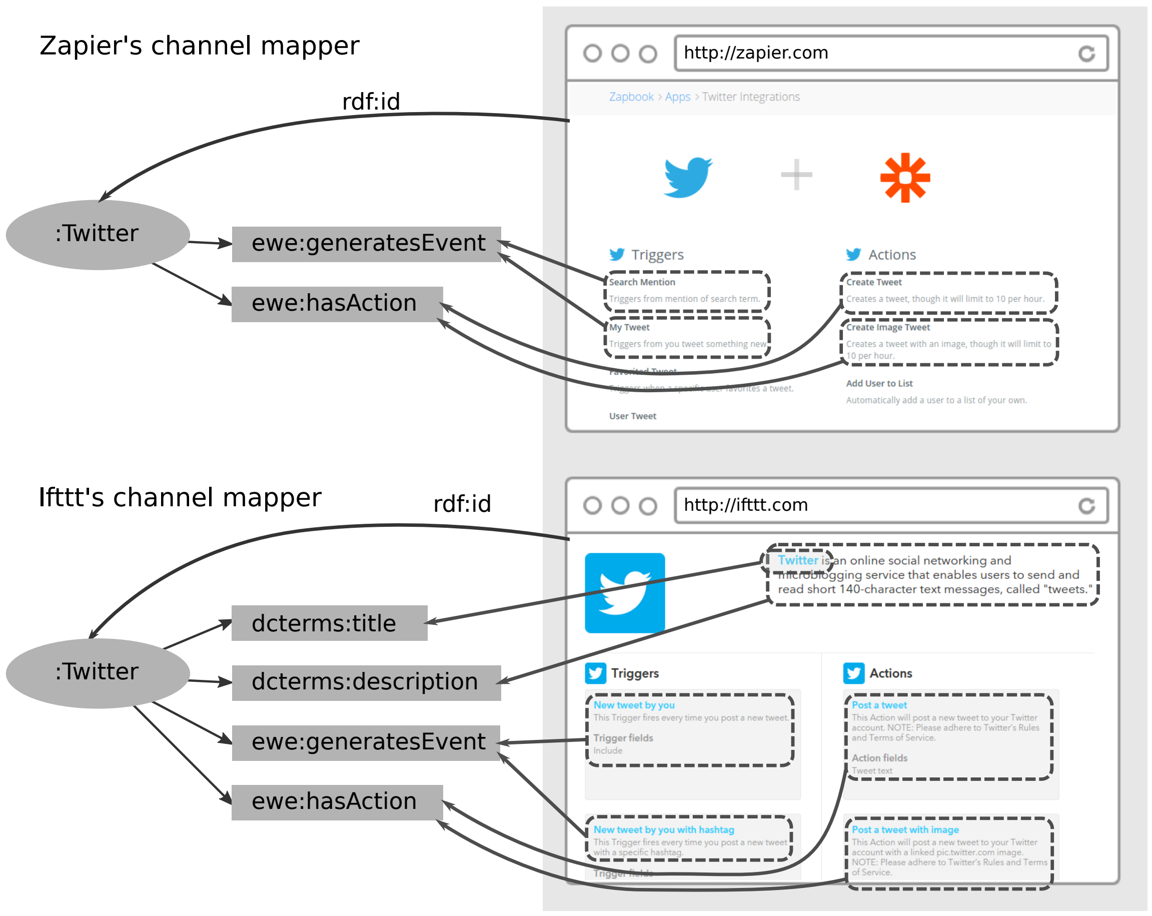
4.3.1 Semantic scraping of TASs websites

Fetching instances constitutes the first step in any ontology learning process. In our case, it consists in extracting the instances from the information about the channels and automations of each TAS's websites, by means of a semantic scraping process. Then, in the following steps these instances will be used to learn the ontologies by applying semantic mapping and clustering based on their semantic similarity.

The existence of multiple commercial TASs that expose information about their channels on their web pages enables this data harvesting process. We say the information about the channels and automations of a TAS is stored in the channel and automation directory: the *channel directory* provides information about all the channels supported by a TAS, and the *automation directory* provides information about all the automation rules created by its users, using the channels from the channel directory. We targeted Ifttt and Zapier, because they are the TASs that integrate greater number of channels, and they have their channel directories available. Zapier's directories are available at `https://zapier.com/zapbook/` and Ifttt's channel and automation directory at `ifttt.com/channels/` and `https://ifttt.com/recipes/` respectively. With this information we populate a *dataset of channels and automations* used in the process of ontology learning. It consists of a set of semantic instances that represent all the channels and automations from the TAS's channel and automation directory. The instances are modelled using the EWE ontology described in the former chapter.

In the EWE evaluation section of the former chapter (section 3.5 of chapter 3), we already made use of this dataset and loaded it into the SPARQL endpoint that we query to show how EWE effectively models the domain data. However, being the instance extraction an important part of the Ontology learning process, we describe it here for convenience.

We have defined a semantic proxy layer on top of the TAS's channel and automation directories. For each TAS, we have defined a mapping between their HTML contents of their channel directory and automation directory and the EWE model. An example of these mapping is shown in Figure 4.2. It presents two snapshots of Ifttt's and Zapier's Web pages, and it highlights the pieces of information used to populate the channel properties. Then we defined several templates to export instances in EWE model to multiple formats such as RDF, N3, or JSON-LD. To define this mapping we use scrappy (Fernández-Villamor et al., 2012), a framework for semantically scrape web resources that lets to separate the data model from the importers (mapping from HTML to the data model) and the exporters (templates to export the data). By using this approach, we reduce the number of mappings required to i) one pair mappings per TAS to transform the html in their channel directory



Search Mention
Triggers from mention of search term.
My Tweet
Triggers from you tweet something new.
Favorited Tweet
Triggers when a specific user favorites a tweet.
User Tweet

Create Tweet
Creates a tweet, though it will limit to 10 per hour.
Create Image Tweet
Creates a tweet with an image, though it will limit to 10 per hour.
Add User to List
Automatically add a user to a list of your own.

Twitter is an online social networking and microblogging service that enables users to send and read short 140-character text messages, called "tweets."

New tweet by you
This Trigger fires every time you post a new tweet.
Trigger fields
Include
New tweet by you with hashtag
This Trigger fires every time you post a new tweet with a specific hashtag.

Post a tweet
This Action will post a new tweet to your Twitter account. NOTE: Please adhere to Twitter's Rules and Terms of Service.
Action fields
Tweet text
Post a tweet with image
This Action will post a new tweet to your Twitter account with a linked pic.twitter.com image. NOTE: Please adhere to Twitter's Rules and Terms of Service.

Figure 4.2: Mapping for scrapping TAS channels from Ifttt and Zapier

and automation directory into the EWE model, and ii) a mapping (or template) to export that common model to each desired format (RDF, N3, etcetera.).

The first exploratory analysis of the extracted data is summarised in Table 4.1. It presents the number of instances extracted from each TAS, showing that Zapier integrates many channels than Ifttt, and in consequence much more events and actions. However, the larger number of Ifttt's users produces more automations (rules) than Zapier's

A deeper inspection of the event and action data, shows that Ifttt's data contains information about event and action parameters, while Zapier's data do not –since that data were not available from Zapier's channel directory. This will stop the ontology learning process from using Zapier's data because the missing information about parameters is important in the following steps of the process.

Next, we searched for the channels that are supported by both TASs –browsing the channel directory in their websites, it may be seen that popular services such as Dropbox, or Twitter are supported by both of them. After a quick tiding up of the channel titles, we

Table 4.1: Number of instances of each type in the dataset

Instance type	**Ifttt**	**Zapier**	**Total**
Channel	195	521	716
Event	725	1532	2257
Action	357	1046	1403
Rule	80353	31501	113073

discovered that 54 channels are shared by Zapier and Ifttt. This is 7.54% of the total number of channels. However, considering the channels involved in the set of rule instances scrapped we calculated that 49.9% of those rules use channels supported by both TAS (22.01% of the Zapier rules and 59.33% of Ifttt rules), i.e. shared channels are much more common in rules than the rest of the channels, and this is so because these channels are more popular among users. These data assist our contribution of developing a common model to transfer automation from one TAS to another, since 49.9% of the rules are liable of being transferred between both TAS fully functional.

The taxonomies of channel categories provided by both TASs highly differ from each other: i) Ifttt assigns each channel a single category, while Zapier tags each channel with multiple categories, and ii) Ifttt defines 15 categories while Zapier defines 27. We run several attempts to deduce a general taxonomy that could be applied to channels from both TAS by simply using the categories assigned to shared channels. However, since both taxonomies are very different and the number of shared channels is small, we obtained no successful outcome. Nonetheless, having a general channel categorisation across different TAS is required to assist users in creating automations, to automatically identify substitutive channels, or to recommend channels and automations. Thus, it leads us to attempt to cluster channels based on the similarity given by the events and actions as we describe in the following step of the ontology learning process.

4.3.2 Calculating channel similarity

The second step consists in calculating the distances between channels that will be used by the clustering algorithm, in the following step. To do so, we propose a method to measure similarity between instances combining semantic structure and linguistic similarity.

Recall that according to EWE model, channels may generate events and provide actions.

Table 4.2: Most frequent parameter titles

Param title	**count**	**Param title**	**count**
Title	155	Tags	92
CreatedAt	151	OccurredAt	63
URL	103	DeviceName	56
Which device?	102	Message	44
Description	95	Caption	37

These events and actions define the structure of the channel, which may be seen as a set of actions and actions. Therefore, the semantic similarity between two channels is given by their events and actions. Likewise, events and actions (for sake of readability we will refer to them as channel elements) have a particular structure that is defined by their parameters (e.g. an event titled "new email received" has three output parameters with the information about the email received: subject, body, and sender). Hence, the semantic similarity between two channel elements is given by their parameters.

On the other hand, Ifttt's and Zapier's channel directory designers have been consistent in naming events, actions and also parameters. As a consequence, elements that have the same title are expected to have the same effect. For instance, both Wordpress and Blogger have an action titled "Create new post", and as expected, in both cases the effect is publishing a post (in Wordpress and Blogger respectively). This consistency in the naming policy also enables using the title of channel elements and parameters to measure (or approximate) its similarity. In the particular case of parameters, harvested data from the former section (section 4.3.1) shows that there is a relatively small set of parameter names that is reused along the list of events and channels. In Table 4.2, the most frequent parameter names are listed.

Considering the example of Blogger's event "New post labelled" and Tumblr's event "New post tagged", both taken from Ifttt channel directory. Both titles are almost synonyms (as tag and label are), so its semantic similarity should be close to a full match. In a further inspection considering the parameters of each event, Blogger's event has *PostTitle, PostUrl, PostContent*, and *Tags* and Tumblr's event has the same param-set but *Tags*, that is exchanged for *Labels*. The pair-wise semantic similarity of these parameters is shown in Table 4.3. Again, its structure is almost identical, so its structural similarity should also be a close match.

Table 4.3: Similarity score between parameters of the example.

	PostTitle	PostUrl	PostContent	Labels
PostTitle	**1**	0.13	0.17	0
PostUrl	0.13	**1**	0.09	0
PostContent	0.17	0.09	**1**	0
Tags	0	0	0	**0.93**

Within these two considerations about the structure of the instances, and the naming policy, we will propose a similarity metric that takes both into account. In particular,

- the similarity between two channels is given by the semantic similarity of their channel elements,
- the similarity between two channel elements is given by a combination of the semantic similarity of their parameter sets, and the linguistic similarity of their titles,
- and the similarity between two parameters is the linguistic similarity of their titles (because parameters do not have structure)

Our first attempt to use a metric that considers this structural similarity is using Jaccard similarity (Jaccard, 1901).

$$Jaccard(A, B) = \frac{|A \cap B|}{|A \cup B|} \tag{4.1}$$

As so, the intersection of sets is the number of elements that share in common (those that are identical), and the union is the total number of distinct elements. However, that approach should be corrected in our case, because there is not a binary comparison between elements, i.e. channel elements are not always equals or not equals, they can also be similar with a certain score. Back to the same example of Blogger's and Tumblr's events. According to binary comparison of elements, parameters "labels" and "tags" are not strictly equals, so they are different elements. In consequence, the similarity score of those two events would be 0.6, value that is far from what it would be expected after what have been formerly discussed.

We propose to correct the way the intersection is calculated, to take into consideration the similarity score of the elements in each set. It consists in accumulating the maximum similarity score between elements of the two sets.

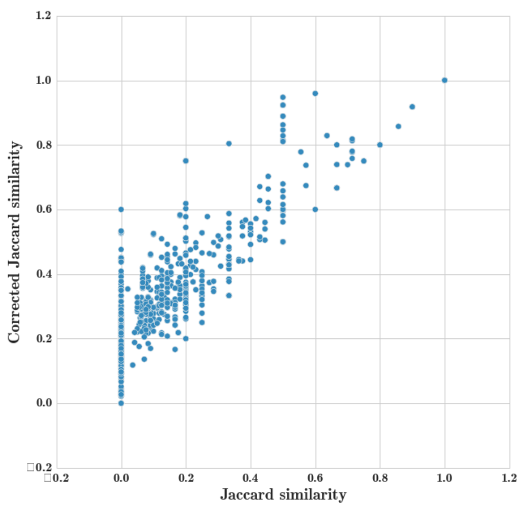


Figure 4.3: Comparison of Jaccard similarity score and its corrected version.

$$Jaccard_{corr}(A, B) = \frac{max\_sim\_acc(A, B)}{|A| + |B| - max\_sim\_acc(A, B)} \tag{4.2}$$

With this correction, and using the similarity values shown in Table 4.3 the value of the accumulated maximum similarity (max_sim_acc) between both element is 3.93, and so the new similarity is 0.96, almost a full match. As shown in Fig. 4.3, where the similarity between all events and actions from the dataset is plotted, the improvement of similarity score in some cases is up to 0.6 points. The implementation of this function, given by the Algorithm 2, searches for pairs of elements from both sets with the maximum similarity. Once a pair is found, they are remove so they cannot match any other element.

Algorithm 2 max_sim_acc

```
**Data**: $E_1$, $E_2$ sets of comparable elements / $|E_1| < |E_2|$
$metric$ similarity metric to use
**Result**: acc_sim: accumulated similarity between elements in both sets
$acc\_sim = 0$
**foreach** $elem\ in\ E_1$ **do**
    $elem2$, $sim$ = max_sim($metric$, elem, $E_2$)
    $elem3$, $sim$ = max_sim($metric$, elem2, $E_1$)
    **if** $elem == elem3$ **then**
        $E_1 = E_1 \setminus \{elem\}$
        $E_2 = E_2 \setminus \{elem2\}$
        $acc\_sim += sim$
**end**
```

Finally, being able to calculate the similarity of channel elements, we can apply the same metric to measure channel similarity. As an outcome, we obtained two tables, i) with the similarity between all channel elements (events and actions separated), and ii) with the similarity of all channels in the dataset. We propose a combination of both, linguistic and structural similarities to compute the element distance tables and channel distance table. The parameter α is used to tune up the weight of each similarity type.

$$\begin{aligned} Sim(A,B) = \alpha \cdot Sim_{ling}(A,B) + (1-\alpha) \cdot Sim_{struct}(A,B) \\ 0 \leqslant \alpha \leqslant 1 \end{aligned} \tag{4.3}$$

Nonetheless, it is important to point out that this similarity metric has one major limitation: it depends on the linguistic similarity of the element's title, which is subject to arbitrary decision of external agents.

Similarity tables contain the similarity measure between each pain of elements in the table. Using a pivot transformation, similarity tables may be seen as a collection of similarity measures between elements. Similarity tables are built using the Algorithm 3, which is based on the former assumption[2].

Algorithm 3 Similarity table compile algorithm

```
Data: $Elems$, set of all elements to compare
$\alpha$ control parameter
Result: $table$ Pair-wise similarity between elements
$table = [\,]$
foreach $elem1$ in $Elems$ do
    foreach $elem2$ in $Elems::tail$ do
        $sim = \alpha \cdot Sim_{ling}(elem1, elem2) + (1-\alpha) \cdot Sim_{struct}(elem1, elem2)$
        $table$.add($elem1$, $elem2$, $sim$)
    end
end
```

We calculate the element distance table with $\alpha = 0.5$, to give equal weight to both components of the similarity measure. This value has been determined in a heuristic inspection using a subset of the channel dataset. A more formal approach to calculate this value is suggested in the future work of this chapter. The channel distance table is calculated with $\alpha = 0$ because using the syntactic similarity between channels has no sense (channel names

[2]For brevity, it uses the expression :: *tail* to address the set of elements that have not yet been processed by the outer loop

are not descriptive). The information in this table may be used to feed recommender systems that suggest channels according to their similarity, or to select substitutive actions and events when a channel is not available. Moreover, in the following steps we will follow this approach to first compute the clusters, and then discover the ontologies.

Recall that, as explained in the former section, the channel dataset does not have information about elements of Zapier's channels. Therefore, it is not possible to compute the similarity of Zapier's channels, and include them on the similarity table. For the rest of the process, only Ifttt data will be used.

4.3.3 Compute channel clusters

The third step consists in computing the channel clusters using the similarity metric, and the similarity tables described in the former step. Each cluster is defined by a set of individuals (the channels) and a set of features that characterise the cluster (the events and actions). In the next step, each cluster will be exported as an ontology using these individuals and features.

We filter the channel similarity table to create a list of all channel pairs with a similarity greater than a threshold Θ_{ch}. For channels pair, we extract the elements they share, and for each of them, we check if there exists a cluster that already has that element among its features. If so, we add the channels to the cluster. Then, all elements that did not match are added as features to the same cluster. However, in the case the channels had been added to more than one cluster (usually when the considered channels share too many elements), those elements are discarded. Finally, in case the channels were not added to any existing channel, we define a new cluster with the elements they share as features. The process is formalised by Algorithm 4.

This strategy has a few advantages from other clustering approaches: i) it does not require the number of cluster a priori, ii) individuals that do not match any cluster are left *unclustered*, iii) individuals may belong to more than one cluster, iv) each cluster provides the features that support that cluster, and v) the algorithm accepts initial clusters, so that when the data from the channel dataset is updated the clusters can be updated too without risk of generating different clusters in each execution.

The clustering process with channels scrapped from Ifttt and a value of $\Theta_{ch} = 0.90$, split 62 of the 195 channels into 22 clusters, leaving 133 channels unassigned to any cluster. The average size (number of individuals) of the clusters is 3.26 channel, and the average number of features per channel 4.13. We selected a value of $\Theta_{ch} = 0.90$ because it is the value at

Algorithm 4 Associative clustering

```
Data: List of channel pairs with the elements they share
Result: Set of clusters with information about the individuals and the features that define
        the cluster
clusters = [ ]
foreach ch1, ch2 in channel_table_row do
    temp_clusters = []
    loose_elems = []
    shared_elems = extract_shared_elems(ch1, ch2)
    foreach elem in shared_elems do
        if elem in clusters.any then
            cluster = clusters.get_cluster_with(elem)
            cluster.add([ch1, ch2])
            temp_clusters.add(cluster)
        else
            loose_elems.add(elem)
    end
    if temp_clusters.size == 1 then
        foreach elem in loose_elems do
            temp_clusters.get().add_feature(elem)
        end
    else if temp_clusters.size == 0 then
        cluster = new Cluster([ch1,ch2], shared_elems)
        clusters.add(cluster)
end
```

which function between the size of the table and Θ has an *elbow*. A summary of the resulting clusters is shown in table 4.4.

As shown, the developed algorithm creates small clusters compared to the number of available items. In particular, more than half the clusters contain only two elements. This behaviour was a design criteria, i.e. finding pairs of channels with meaningful relations is the pursued outcome, since the clustering purpose is to discover relations among channels that will be exported into the new ontologies. We will deeply discuss these data in the evaluation (section 4.4) where we will also assess the sense of these clusters.

Table 4.4: Overview of the cluster sizes, number of features and mean similarity between channels of the clustering outcome with Ifttt data.

cluster	size	features	similarity	cluster	size	features	similarity
0	5	10	0.3110	12	5	2	1.0000
1	6	5	0.2950	13	2	1	0.6187
2	7	15	0.2230	14	6	7	0.3045
3	2	1	0.6455	15	2	3	1.0000
4	2	1	0.6568	16	6	7	0.3874
5	5	8	0.5902	17	2	2	1.0000
6	4	2	0.4105	18	3	8	0.5019
7	2	2	0.5367	19	2	2	0.6421
8	2	2	0.5655	20	2	4	0.7920
9	2	2	0.5410	21	2	6	0.7142
10	2	2	0.7708	22	2	1	0.6363
11	2	2	0.7165				

4.3.4 Generate vocabularies

Finally, the clusters are used to generate the vocabularies by means of templates to export the ontology in N3 or RDF/XML formats. Each vocabulary contains a main class representing the channel that aggregates the properties of the cluster channels. For instance, in the case of a cluster that contains blogging related channels like Wordpress, Tumbler or Blogger, it would be the *blogging:BlogChannel* class. Each vocabulary also provides a set of classes that represent the events and actions that the cluster channels share. Carrying on with the same example, Wordpress, Tumbler and Blogger share the actions with title "Create a post" and "Create a photo post", so the vocabulary includes class definitions for actions *blogging:CreateAPost* and *CreateAPhoyoPost*. Listing 4.1 shows an excerpt of the blogging ontology used in the example.

Listing 4.1: Excerpt of the blogging ontology learnt.

```
@prefix ewe: <http://gsi.dit.upm.es/ontologies/ewe/ns/> .
@prefix blogging: <http://gsi.dit.upm.es/ontologies/ewe/blogging/ns/> .
@prefix ifttt: <http://ifttt.com/channels/> .

# Channel definition
blogging:BlogChannel a owl:Class ;
  rdfs:label "BlogChannel"@en ;
  rdfs:subClassOf ewe:Channel .

{ ?channel a ifttt:Wordpress} => { ?channel a blogging:BlogChannel } .
{ ?channel a ifttt:Tumblr} => { ?channel a blogging:BlogChannel } .
{ ?channel a ifttt:Blogger} => { ?channel a blogging:BlogChannel } .

# Events definition
blogging:NewBlogPost a owl:Class ;
  rdfs:label "New blog post"@en ;
  rdfs:comment "A new blog post was created"@en ;
  rdfs:comment "Fired with new posts"@en ;
  rdfs:subclassOf ewe:Event ;
  ewe:generatedBy blogging:BlogChannel .

{ ?event a ifttt:NewBlogPost } => { ?event a blogging:NewBlogPost } .
{ ?event a blogging:NewBlogPost ; ewe:generatedBy ?channel }
=>
{ ?channel a blogging:BlogChannel}

[...]

# Actions definition
blogging:CreateAPost a owl:Class ;
  rdfs:label "Create a Post"@en ;
  rdfs:comment "Create a regular post"@en ;
  rdfs:subclassOf ewe:Action ;
  ewe:generatedBy blogging:BlogChannel .

{ ?action a ifttt:CreateAPost } => { ?action a blogging:CreateAPost } .
{ ?action a blogging:CreateAPost ; ewe:providedBy ?channel }
```

```
=>
{ ?channel a blogging:BlogChannel}

[...]
```

In addition to the classes, each vocabulary contains a set of assertions written in N3 (Berners-Lee and Connolly, 2011)[3] to match instances of the original classes with the new classes in the vocabulary. In the assertions shown in the Listing 4.1, instances of classes *ifttt:Wordpress*, *ifttt:Tumblr*, and *ifttt:Blogger* are set to be of type *blogging:BlogChannel.* And a similar process is applied to the events and actions. Moreover, it also includes a rule to indicate that the channel that generates the events and actions of this vocabulary must be a blogging:BlogChannel.

By means of these assertions, semantic TASs and other applications may benefit from these vocabularies, even when the instances the work with are not annotated using these vocabularies, because they would be inferred. For instance, when a software agent is extracting the information from a semantic TAS that exposes its channel directory using EWE but it does not include any other vocabulary. The agent extracts the instances but needs extra semantic information. Using this methodology, the vocabularies are automatically generated, and the previously extracted instances are enhanced, once the inferences are performed.

The vocabularies also provide mappings to external resources in the LOD cloud. For instance, the posts generated by actions from the BloggingChannel are of type *sioc:Post*, and have the properties of the SIOC ontology (Berrueta et al., 2010). This is also performed using additional assertions as shown in Listing 4.2. There, events generated by instances of class blogging:BlogChannel are annotated using the properties *dc:title*, *sioc:content* and *sioc:topic*. With these rules we are overloading the instances to provide as much semantic information as possible. The mappings to the external vocabularies are extracted from the Linked Open Vocabularies (LOV) database, using the provided API[4].

Listing 4.2: Example of property mapping in the blogging ontology learnt.

```
@prefix ewe: <http://gsi.dit.upm.es/ontologies/ewe/ns/> .
@prefix blogging: <http://gsi.dit.upm.es/ontologies/ewe/blogging/ns/> .
@prefix ifttt: <http://ifttt.com/channels/> .
@prefix sioc: <http://purl.org/dc/terms/> .
@prefix dc: <http://purl.org/dc/terms/> .

{ ?event ewe:generatedBy ?channel .
```

[3]This is a more readable alternative to OWL class inferences (Bechhofer, 2003)
[4]`http://lov.okfn.org/dataset/lov/api`

```
  ?channel a blogging:BlogChannel .
  ?event ifttt:title ?title .
  ?event ifttt:body ?body .
  ?event ifttt:category ?category .}
=>
{ ?event a sioc:Post ;
  ?event dc:title ?title .
  ?event sioc:content ?body .
  ?event sioc:topic [ rdfs:label ?category] } .
```

This step requires manual post processing to i) give a meaningful namespace to each ontology, and ii) check the proposed mappings of properties to external ontologies are correct. Nonetheless, these adjustments are not essential to obtain the vocabularies. The alternative is to obtain vocabularies with an alphanumeric namespace that identify them (this is a common solution in programs that convert from XML to N3, and in SPARQL endpoints), and a non filtered list of property mappings that may be inaccurate, which is of no harm in an open world assumption.

4.4 Evaluation

In this section we propose an evaluation for the methodology described in the former section. Although the evaluation of ontology learning procedures is still an open problem, there is already some work in this direction. Shamsfard and Barforoush (2003) present two basic approaches for evaluating these systems: the evaluation of the underlying learning methods and the evaluation of the learned ontology. However, because of the difficulty concerning the measurement of the correctness of the learning procedure,the former approach is less addressed. According to Dellschaft and Staab (2006), the resulting ontologies can be compared by evaluating them in a running application, a posteriori evaluation by experts, or evaluation by comparison of learned results against a predefined gold standard.

In our case, we chose to perform an evaluation of the learnt ontologies by an expert. Manual evaluation has advantages, since experts are supposed to know the concepts and their relationships in their domain of expertise and, therefore, they are supposedly able to tell whether a given ontology represents the domain or not. However, it also has their drawbacks. For instance manual ontology evaluation is subjective and time consuming, and it is not feasible for large-scale evaluations (Dellschaft and Staab, 2006).

4.4.1 Evaluation metrics

To communicate the results of the evaluation, we calculate the precision and recall of the process measured at two levels: i) at a dataset level, evaluating whether the meaningful vocabularies are discovered, and ii) at a vocabulary level, evaluating whether the composition of each vocabulary is correct and complete. At a dataset level, *recall* is the fraction of meaningful vocabularies identified among all those in the dataset (equation 4.4), and *precision* is the fraction of correctly identified vocabularies (equation 4.5). The following section will describe how the terms from the equations are computed.

$$Recall_{dataset} = \frac{\mid correctly\_generated\_vocabularies \mid}{\mid all\_discoverable\_vocabularies \mid} \tag{4.4}$$

$$Precision_{dataset} = \frac{\mid correctly\_generated\_vocabularies \mid}{\mid all\_generated\_vocabularies \mid} \tag{4.5}$$

At a vocabulary level, recall is the fraction of identified channels that belongs to the vocabulary (equation 4.6), i.e. leaving out of the vocabulary channels that belong to it decreases the recall. Precision is the fraction of correctly identified channels term (equation 4.7), i.e. including in the vocabulary channels that do not belong to it decreases the precision.

$$Recall_{vocabulary} = \frac{\mid correctly\_identified\_channels \mid}{\mid all\_belonging\_channels \mid} \tag{4.6}$$

$$Precision_{vocabulary} = \frac{\mid correctly\_identified\_channels \mid}{\mid all\_channels\_included \mid} \tag{4.7}$$

To combine both measures in a single score, we use their harmonic mean, commonly known as the F-score (equation 4.8):

$$F = 2 \cdot \frac{precision \cdot recall}{precision + recall} \tag{4.8}$$

4.4.2 Expert assessment and results

The process consists on a manual assessment by a domain expert who evaluates the two levels defined in the former section.

At the vocabulary level, the initial number of channels in the vocabulary is the *all channels included* term from the equation 4.7. The expert removes from the vocabulary the channels that are misplaced, i.e. those that are not similar to the rest of the channels.

The resulting size of the vocabulary is the *correctly identified channels* term (equations 4.6 and 4.7). Then, the expert adds those that, not being included by the automatic learning process, are similar to the rest of the channels in the vocabulary. The number of resulting channels in the vocabulary is the *all belonging channels* term (equation 4.6).

At the dataset level, the initial number of vocabularies automatically generated is the *all generated vocabularies* term from the formula 4.5. The expert studies which vocabularies present a set of channels which offer a similar service, e.g. blogging channels, email channels, activity trackers, etcetera. Those that does not meet these criteria are removed. This is usually the case of small clusters where two or three channels are put together because they share an event/action with a similar structure or title, but they do not offer a similar service. The number of resulting channels is the *correctly generated vocabularies* term (equations 4.4 and 4.5). Finally, the expert analyses the channel directory of the TAS to figure out if there is any other group of channels that offer a similar service, that share events or actions with a similar structure, and create those new vocabularies. This final number of vocabularies is the *all discoverable vocabularies* term (equation 4.4).

Despite the subjects that performed the evaluation were domain experts, they had to consult the channel directory of the TAS frequently, because knowing which channels are supported is out of their expertise. Since manual evaluation is a subjective process, we reduce the effect of subjectivity we repeated the evaluation with two different experts.

Out of the 23 vocabularies generate by the ontology learning process, we identified 18 of them are meaningful vocabularies, while the other 5 are either vocabularies that should be included in other vocabularies, duplicate vocabularies or meaningless vocabularies. We assign a name to these vocabularies (as pointed out in section 4.3.4) as presented in Table 4.6. In addition, one meaningful vocabulary was not identified in the process. With these data, the resulting precision is 94.74%, the recall 78.26% and the F-score 85.71%.

At a vocabulary level, each single vocabulary was assessed annotating the missing and exceeding channels. The computed values of precision, recall and F-score are shown in 4.6. The average precision considering all vocabularies is 96.03%, the average recall 85.56% and the average F-score 87.93%.

These results show that the resulting ontologies effectively model the specific channel domains, and therefore the ontology learning methodology this chapter contributes with effectively generates them with little human supervision.

Another advantage of this approach is related to reusability of the resulting ontologies. Due to the high complexity and error prone of some manually developed ontologies or the

Table 4.5: Dataset level precision, recall and F-score

Measure	Precision	Recall	F-Score
Dataset level metrics	94.74%	78.26%	85.71%
Average of vocabulary level metrics	96.03%	85.56%	87.93%

high specificity of the automatically generated (Sabou et al., 2005; Shamsfard and Barforoush, 2003), developers usually prefer to build new ones causing a duplicity of models in the state of the art. The ontologies produced by our approach are compact and simple, so that they are easy to understand, reuse and extend.

Table 4.6: Vocabulary level precision, recall and F-score

Vocabulary title	Channels	Features	Precision	Recall	F-Score
Social networks	5	11	100%	100%	100%
News	5	2	100%	100%	100%
Blogging	6	7	42.86%	100%	60%
Activity trackers	3	8	100%	100%	100%
Picture sharing	7	15	85.71%	100%	92.31%
Cloud Storage	2	2	100%	66.67%	80%
Geolocalized photos	2	1	100%	40%	57.14%
Favouriting	2	1	100%	100%	100%
Bookmarking	5	8	100%	100%	100%
Video feed	3	4	100%	66.67%	80%
Positioning systems	2	3	100%	100%	100%
Connected Lighting	6	7	100%	100%	100%
Connected Air-conditioning	2	2	100%	100%	100%
Connected Automotive	2	1	100%	50%	66.67%
Meteorological stations	6	5	100%	100%	100%
Visual notifications	2	2	100%	66.67%	80%
Feeds saving	2	1	100%	100%	100%
Messaging	2	1	100%	50%	66.67%

4.5 Discussion and Outlook

In this chapter, we complement the existing semantic reference model EWE developed in Chapter 3, with a set of automatically learnt vocabularies that may be used for automatic channel discovery, and for recommending substitutive channels in task automation composition.

The first contribution of this chapter, was to harvest data of channels and automations from the channel and automation directory of existing TASs, describe them using the common semantic model, and present them as a dataset fed from different sources. This is relevant information to be used by software agents and users to consult information about the TASs.

The second contribution was to establish a similarity metric between channels, suitable when they come from different sources. We define the structure of a channel as the set of events and actions it provides. The resulting metric takes into consideration this structure, the parameters that each event and channel exposes, and the title used to identify/describe them. We discovered that the way different events or actions are titled, is a good measure of their similarity. However, this is also, a limitation because the similarity relies on an arbitrary decision made by external agents. We balanced the weight of this similarity with that provided by the set of parameters, which also measures its compatibility in terms of swapping both evens/actions. This similarity is presented as a correction of the Jaccard similarity, and it has been evaluated in the former dataset concluding the results of the corrected similarity are more accurate, in the domain under study, than those obtained using the original similarity.

The third contribution was to design a process to automatically learn ontologies from the dataset of channels using the similarity metric described. It clusters the similar channels, extract the features they share and export that as ontologies. In the process, a channel categorisation arise. This is different from the categorisation some TAS offer, since it is not based on those taxonomies, but in the similarity of channels. It has been proved that this clustering algorithm presents advantages in this scenario such as, i) it does not require to number of cluster a priori, ii) individuals that do not match any cluster are left unclustered, iii) individuals may belong to more than one cluster, iv) each cluster provides the features that support that cluster, and v) the algorithm accepts initial clusters, so that when the data from the channel dataset is updated the clusters can be updated too without risk of generating different clusters in each execution. Finally, we used the harvested instances, the similarity metric and the clustering algorithm to automatically learn ontologies. The results have been evaluated at a dataset level, and at an ontology level obtaining a combined F-score of 86.82%.

As future work, we intend to evaluate the resulting ontologies against other ontology learning approaches that use different clustering algorithms, with the goal of refining the learning process, aiming to minimise the number of non meaningful ontologies produced, i.e. those that make sense from the point of view of the structure of the channels involved, but that they do not represent a category of channels. Moreover, in the evaluation process we will take into account more TAS, extracting data from more different sources, so increasing the size of the dataset. We aim to make the process self-adaptive to the source of the date, by harvesting more semantic relations, the process will take into account the availability of data, the naming policy, or the local categorisation used by each source.

On the other hand, the extracted data will be used to develop a metadirectory of TAS channels, that user may consult to check which channels are supported by each TAS, which channels are similar to other in terms of the function they perform or the structure they offer, how two events or actions may be swapped, etc. In a next iteration of this metadirectory, it will include the possibility of orchestrate automations and export them using EWE format.

Finally, with this information, a comprehensive data analysis of the automation rules may be performed to extract different profiles of users and automations as well.

CHAPTER 5

Personal Agent Architecture for Task Automation in the Web of Data

This chapter describes a personal agent architecture for TASs that brings intelligence into task automations. It presents a few case studies to show the benefits of applying agents in a TAS scenario, and helps the reader to have a clear understanding of the possible tasks the agent might develop to assist the users.

5.1 Introduction

In the former chapters, we addressed the problem of semantically describing task automations and their components. We provided a specification of a Semantic Reference Model, and designed a method for automatically learn vocabularies that populate an automation dataset using the aforementioned model.

Agent systems can exploit the potential of task automations –benefiting from the semantic technologies used– by facilitating the users the creation and management of these automations. In particular, personal agents can use additional personal and contextual information to manage task automations on behalf of users. However, this approach requires that the agent and the TAS share the information of personal streams (web services, smartphones, sensors).

The need of immediacy and interaction with services is a recurring topic addressed in the state of the art in different application fields, such as connected home (Costa et al., 2012; Yang, 2013), collaborative Web scenarios (Jung, 2011), travel assistance (Yueh et al., 2007), etcetera. Unfortunately, current agent platforms do not provide any standardised mechanisms to integrate external sources. The challenge consists in proposing a flexible and modular architecture that features seamlessly interactions with a variety of streams, regardless of their nature. For instance, the integration of sensors and actuators typically requires extending the basic agent architecture and a deep understanding of its implementation. Furthermore, this challenge is not exclusive of agent systems, TASs suffer from it too.

Thus, this chapter briefly reviews the state of the art of personal agents, and extends a former work by Rada et al. (2014), modifying the architecture proposed to address the TAS scenario. The architecture, called Modular Architecture for Intelligent Agents (MAIA), provides an event-based perspective and a modular design. This chapter also explains how this architecture provides the TASs connectivity to multiple data streams, sharing the adapters and connectors with the agent architecture.

This chapter is structured as follows: Section 5.2 introduces several concepts of personal assistants and agent platforms that are relevant to this chapter; Section 5.3 presents an overview of the proposed architecture called, describing in detail the function of each module, and how the communication is orchestrated; Section 5.4 presents a list of proactive agent behaviours to assist the user in a TAS scenario; Section 5.5 describes a case study where the personal agent help the users to manage their automation rules, showing how all the communication is handled by the proposed architecture. Finally, in Section 5.6 we present the conclusions to this chapter.

5.2 Background

In this section, we introduce some of the background technologies and efforts made in the state of the art that are related to the research described in this chapter. It includes personal agent –classification and behaviours–, and agent architectures.

5.2.1 Personal Assistants

Personal Assistants (PA) are agents that can represent individuals in a certain environment: the Web, an action, when scheduling a meeting, etcetera. They help users in their day-to-day activities, especially those involving information retrieval, negotiation, or coordination. A personal assistant might schedule a meeting and then, based on the meeting location, find the nearest baby sitting service (Huhns and Singh, 1998; Olsen and Malizia, 2011).

With the rise of smartphones, PA shifted from the Web to mobile devices, which provides several advantages such as immediacy, ubiquity or access to enhanced user information. As smartphone PA became popular, all smartphone OS developed their own pre-installed PA. These are Apples' Siri, Google Now or Microsoft's Cortana. There is also a growing trend in including Natural Language Interface (NLI) in PA, which the three former examples already implement. However, PAs spread beyond smartphones and reached connected homes and IoT. There are multiple projects on this topic that provide a physical device that connects to the Internet and provides a NLI to assist the user with common queries about weather, user's agenda, movies, etcetera. The most popular is Amazon Echo, which nonetheless is still work in progress (Dempsey, 2015).

5.2.1.1 Personal assistant classification

According to the task they perform, Personal Agents can be classified into five main categories: personal information management agents, purchasing and trading agents, task and time management agents, reminder agents, and recommender and filtering agents. Some authors suggest an additional category named decision-making agents (Ohmukai et al., 2003), that may also address some agents of the former categories. *Personal Information Management agents* (Vassileva, 2008) help users in acquiring, organising, retrieving, and processing information in their personal spaces, e.g. spreadsheets, email messages, contact lists, to-do lists, and web bookmarks, but it also includes physical items.

The objective of *purchasing agents* is assisting the user in the shopping process, by filtering the best item to purchase according to a search criteria, and if the case execute the purchase. Purchasing agents are heavily driven by user profiles, e.g. in some cases being a trustworthy seller or providing a fast delivery may take precedence over price. Similarly, interpersonal opinions have been found to have a significant influence (LaTour and Henthorne, 2015). Taking the decision of making a purchase is delicate and usually involves several goals to attain he best profit. Choices made about how to pursue each of these goals may well result in a set of intentions which are conflicting, thus these agents should be able to reason about and modify its set of intentions to take account of such issues (Shapiro et al., 2012). In this category, travel agents are the most common example, since the price of the trip is the prevailing factor when travelling, as opposed to other kinds of sales. On the other side, we find the *trading agents*, a more specialised kind of agent which has to face the inner complexity of the stock market, and have a heavy communication component.

Time management refers to the process of helping a user manage actual and potential temporal commitments. *Task management* involves the planning, execution, and oversight of tasks. These tasks may be personally imposed or they derived from her responsibilities. Theoretically, these agents organise the tasks and duties of the user in order to make her more efficient. Task management agents fetch information about every task (its deadline, the priority, the workload, etc.) and use optimisation algorithms to generate possible schedules. This is a complex type of agent, due to the large number of factors taken into account when planning the schedule of a user, the need for such planning to change based on contingencies that may arise, and the most important: people have their own standards for judging and scheduling (Ohmukai et al., 2003). An important point in every user's working agenda are meetings, since they are very important and common events. Therefore, many time management personal agents focus on managing commitments and collaborative decision making (Indiramma and Anandakumar, 2008). They are designed to enhance user participation in a meeting through mechanisms that track the topics that are discussed, the participants' positions, and any resultant decisions, providing tools that improve decision making. Recently, social task networks(Gil et al., 2010) have been proposed as a social approach to task management (Indiramma and Anandakumar, 2008), taking into account that tools such as to-do items usually show relationships among users' tasks, their social network and web resources.

Reminder agents are a fairly simple type of persona assistants which check the users data-streams in order to remind them about important events. Google Calendar is a great example of reminder agent since it tracks all meetings and task deadlines and allows the user to set custom alerts. Other implementations make use of artificial intelligence techniques to determine what action should be performed, depending on the nature of the event. For instance, depending on the user location and event venue, the reminder will appear soon enough to allow the user to attend the meeting on time (Coronado et al., 2014b).

Recommender agents are an important and widespread type of agents. The key issue in making recommendations is extracting the user's profile (Montaner, 2003). These agents appear on the Internet associated with other different agents, since almost all types of agents mentioned above use a recommender agent as well for some purpose. Some different recommender agents according to their application domain are e-commerce, e-mail filtering or web, movie, travel and news recommender, etcetera.

5.2.1.2 Personal assistant behaviours

The literature surveyed by Lin and Carley (1993) distinguishes between two types of agent behaviour: reactive and proactive, to which some authors add a third type, a social behaviour (Wooldridge and Jennings, 1995). A *proactive agent* asks for information, reads it, then makes a decision based on the available information, and passes on the decision. Unlike the proactive agent, a reactive agent will not make a decision and pass it on unless being requested by the user. Consequently, when an agent with proactive behaviour performs a task without having been instructed to do so the efficiency the user perceives is clearly superior. Apparently, the assistant completed its task instantly, even when it lasted quite some time. On the other hand, it is possible that the work done by the agent is not valuable, but its availability without having been requested gives a certain added value. (Gruber, 2009). Thus, what makes agents so powerful is their proactive behaviour.

Proactive behaviour is also seen as an essential characteristic of autonomous and semi-autonomous agents (Norman, 1994). As said, the goal of personal agents is helping the user on completing a task. Agents may aid the users directly by performing tasks on their behalf or in conjunction with them (Grosz and Kraus, 1996), and indirectly through actions such as providing context information, minimising interruptions, and offering suggestions and reminders (Czerwinski et al., 2004).

Yorke-Smith et al. (2012) split proactive behaviour into two types. The first type, called *task-focused proactivity*, involves providing assistance for a task that the user either is already performing or is committed to performing; assistance takes the form of adopting or enabling some associated subtasks. For instance, Task-focused proactivity behaviour collects background information in support of a scheduled meeting. The second type of proactive behaviour, called *utility-focused proactivity*, involves assistance related to helping the user generally with her set of tasks, rather than contributing directly to a specific current task. An example of this type occurs when and assistant takes the initiative to recommend transferring a paper review task in response to the detection of high workload levels. This action is triggered not by a motivation to assist with any individual task on the user's to-do list, but rather in response to a higher-level motivation (namely, workload balancing).

5.2.2 Agent Architectures

In the 1990s, research interest was focused on the investigation of architectural issues raised by three influential threads of agent research (i.e. reactive agents, deliberative agents and interacting agents), as collected in the excellent survey by Müller (1999).

Software agent platforms are usually specialised in a particular agent architecture. For instance, most platforms for deliberative agents have adopted the Belief-Desire-Intention (BDI) model, e.g. Jadex (Pokahr and Braubach, 2009), Jack (Wallis et al., 2002) or Jason (Bordini and Hübner, 2005), while the most popular agent platform for interacting agents, Jade (Bellifemine et al., 2007), is based on FIPA (Steiner, 1998). Some of these platforms provide facilities to combine reasoning and interacting features, such as Jadex or Jason, which can be integrated with Jade.

The BDI architecture defined by Rao et al. (1995) is based on the original model proposed by Bratman for modelling human reasoning (Bratman, 1987). The BDI abstract architecture models human-like reasoning by capturing the mentalist notions of belief, desire and intention, which are processed according to a generic interpreter. This interpreter assumes that events are atomic and recognised after they have occurred.

Traditionally, both messages and precepts have been managed in the same interpretation cycle, as both are considered forms of external events. Consequently, most agent implementations mix reasoning processes with the communication logic and make them hard to reuse, debug and develop. Recently, several works such as ACRE (Lillis, 2012) and Alfonso et al. (2011) have proposed to delegate conversation management in a specific module external to the agent reasoning process. The interaction between these two modules is done through actions and perceptions. The reasoning module can reason about the outcomes of every conversation through a set of predefined perceptions, and then execute several actions to manage the status of those conversations (e.g. cancelling, forgetting or retrying a conversation).

Several works have proposed different mechanisms for integrating agents and web services, as surveyed in (Greenwood et al., 2007). The existing solutions provide mappings between addressing and messaging schemes in web services and agent systems. They are implemented using a gateway that publishes web service descriptions into FIPA's directory facilitator and vice versa. Nevertheless, this solution is rather complex. For some applications, a more lightweight solution that integrates intelligent agents and web services would be desirable.

5.3 Modular Architecture for Intelligent Agents and Task Automation

This section discusses the main design choices behind the Modular Architecture for Intelligent Agents and Task Automation, and presents the main modules of the architecture (Figure 5.1), focusing on the relationship between them. Each module is described in greater detail in a separate subsection, including the underlying communication mechanism, and the interconnection with the TASs components. This architecture is a continuation of a former work (Rada et al., 2014), under the name of MAIA, and it was designed with the aim of being an architecture that provides transparent access to data-streams and web services. Thus, it is the perfect solution to integrate agents systems with TASs.

This architecture introduces a few elements in the Reference Architecture for TAS presented in Chapter 2. These are the Evented Web Bus, the Agent Bus and the Event Manager. The driving forces behind it are modularity and loose coupling, i.e. the architecture is designed to allow adding new modules that expand the capabilities of the system. External modules are connected to one of the two buses, either directly or through an adapter (Section 5.3.1). This isolates every components that could be plugged and unplugged independently.

The *Evented Web Bus* 5.3.2 connects to external data-streams (web services, connected devices, etcetera). e.g., an email server or a sensor network. In general, an adapter will be needed. The *Agent Bus* 5.3.2 connects to the modules that are closely related to a typical agent (BDI platform, sensors, actuators, etc.) The *Agent Bus* 5.3.2 provides an entry point to the agent platforms, but also connects the TAS clients with the rest of the system. The *Event Manager* mediates between both buses, providing extra services to the as described in Section 5.3.3. These services will have an important role in the development of BDI agents. In Figure 5.1, LOD linked and Semantic Reasoner were included, but there are much more services available.

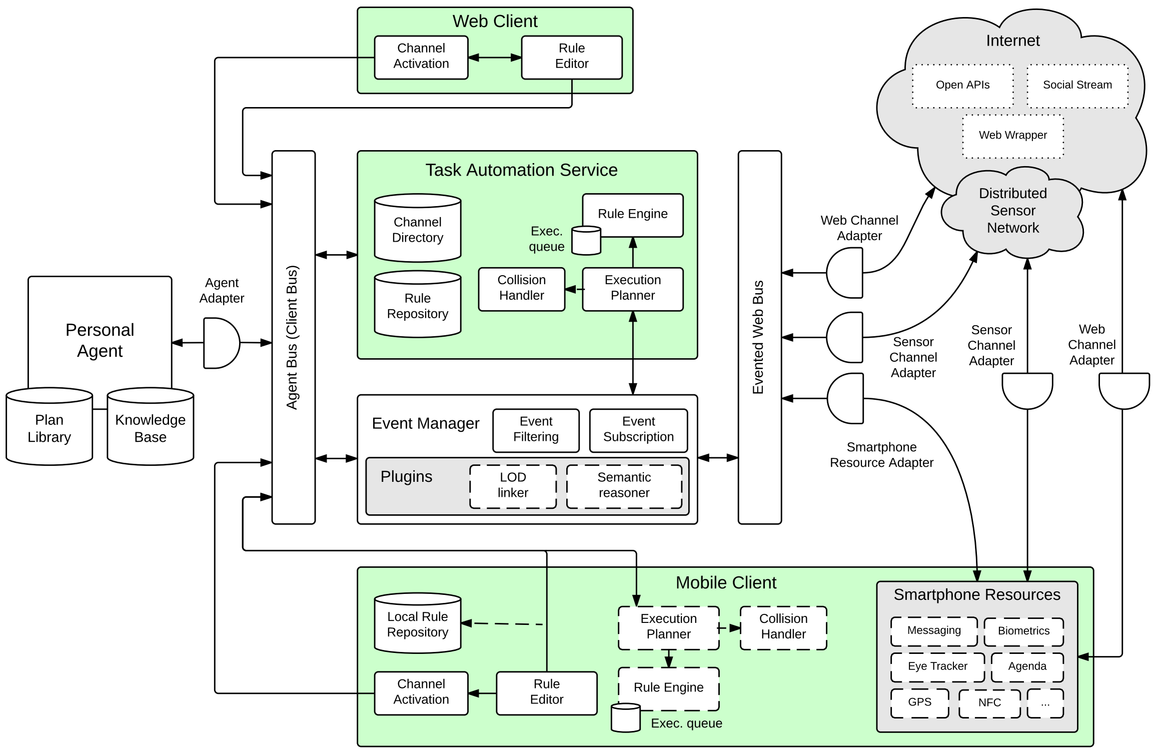


Figure 5.1: Modular Architecture for Intelligent Agents and Task Automation.

5.3.1 Adapters

To be able to connect to any of the buses a module must communicate via events and use one of the protocols that its bus implements. Unfortunately, not all systems are natively evented. Even when they are, they do not always follow the events format (explained by Rada et al. (2014)) or use the same protocol as the bus.

An Adapter is a piece of software that mediates between such systems and the rest of the modules. In the best case scenario, which is that of software that is already event oriented, the adaptation process is as simple as translating event formats on the fly and dealing with protocol differences. In the worst case scenario, deeper changes in the software itself might be needed.

We group the adapters in two categories according to the level of integration they provide: basic adapters and Agent Adapters. Basic adapters make the features of an external service or module available to the rest of the modules. Agent Adapters also make the advanced services provided by the Event Manager available to the module in question.

In essence, basic adapters simply add sources of information or interaction with external services, whereas an Agent Adapter connects to a module with more complex logic.

These adapters take care of: connecting with the Event Manager; translating event formats back and forth; generating events that are compliant with the supported format, and storing events for later consumption. Every adapter that connects to the Evented Web Bus is a basic adapter.

Agent Adapters are the interface between an agent system, typically an Agent Platform, and the Agent Bus. The role of these agent systems is to implement the logic of the final application, adding intelligence to the system and communicating to the different modules. The Event Manager provides several services to make it easier to perform certain common actions or simply delegate tasks that would otherwise be done by the agent. Thus, an Agent Adapter should integrate these services in the agent platform.

The design and features of the Agent Adapter highly depend on the target Agent Platform, its internals and the programming interface it offers. Hence, we will focus on the development of an adapter for Jason. Nevertheless, most of the concepts herein are general and apply to other Agent Platforms.

We identified three main challenges in the adaptation process. The first one consisted in communicating with the platform itself, and its individual agents. The second one was translating events to beliefs, in this case Jason beliefs. Lastly, there needs to be a way to use the extra services provided by the Event Manager from within any Jason agent.

Every agent within Jason has its own knowledge database, which is populated by data from the different sources. To be able to actually modify the perceptions of the agents, a custom Jason Environment is needed, along with an ad-hoc model for this scenario. By modifying the basic Jason Environment we are able to control not only the sources through which new information is added, but the life cycle of such information.

More precisely, the custom model follows the data inbox concept, the same as regular mailboxes. All information received by the agent is volatile, and will be discarded after it is fetched. Should the agent find the information interesting or necessary for the future, it will save it as beliefs in its permanent knowledge database.

Using these data boxes it is rather easy to integrate our Java code and our agents in AgentSpeak. A special function allows any Java method to send information to any certain agent, and any Java function can be wrapped and made available to the agents in the platform.

Apart from the modifications explained above, events themselves need to be converted to beliefs internally. For this purpose, we created the libraries to translate a subset of the JSON notation to beliefs and vice versa. Unfortunately, the limited syntax of beliefs makes it impossible to perform a complete mapping.

Lastly, it is important to note that every agent should subscribe only to those events that are relevant to its functioning, and to avoid permanently storing them. Otherwise, we risk overloading the agents with too many facts, which hinders the reasoning process and might lead to undesired behaviours.

5.3.2 The Agent Bus and the Evented Web Bus

The role of the buses is to communicate external nodes with the Event Manager. The architecture differentiates between two buses: the Agent Bus, which connects to the high-level components of the agent, and the Evented Web Bus, to connect to external services. For instance, the Agent Platform, the User-Interface, and the Communication Manager would be connected to the Agent Bus. In contrast, micro-blogging or web services would interact with the Evented Web Bus.

The reason behind this separation is twofold: it draws a clear line between the access to services and logic, and it allows the Event Manager to provide additional high level services only to the Agent Bus. Most of these services, which will be discussed in the next section, are focused on the development of personal agents that interact with social networks.

5.3.3 Event Manager

The Event Manager is the bridge between the two buses. One of its roles is to exchange events between them, making Evented Web and sensory information available to agents and forwarding requests from agents to services. However, such information is usually verbose and frequent. Most of the times it is redundant or not critical. In contrast, the communication among agents or between agents and the user interface are usually more critical and sensitive to delays. As a consequence, the exchange between both buses has to be controlled. That is the role of the Event Manager. In addition to the plain message passing provided by the buses, it adds event filtering, event subscription, and store. In addition to plain message passing provided by the buses, the event manager adds event filtering, event subscription, and store and forward. Event Filtering allows selecting only the relevant events in each situation and for each module. By using Event Subscription modules can indicate their interest in certain kind of event which they wish to receive. These subscriptions can be used for event filtering as well. Store and Forward means that modules can receive the events they subscribed to and that were sent while they were disconnected. It also means that events will be saved until they can be forwarded to a module. Without it, an overloaded module would not be able to consume all the events sent to it, which might then be discarded.

Besides controlling the flow of events between the different modules, it complements the Agent Bus by providing higher level functions that are not present in it, these are the so-called plugins 5.1. The Event Manager provides several useful services for the development of personal agents. Namely, these services are: Identity, Event Based Task Automation, Location, Semantic Information, Social Networks, Calendar and Transactions.

The Identity Service allows agents to define virtual identities. These identities can be linked to the rest of the services. For instance, an identity can be linked to several calendars and social networks. These identities are defined via FOAF (Brickley and Miller, 2014). Each identity has a unique ID that can be used to subscribe to the events from the sources linked to it. The Event Based Task Automation offers the option for agents to delegate actions to the Event Manager. These actions will be fired by a certain event, and their result will be another event.

The Social Network service homogenises the connection and interaction with different social networks. Social networks are an important part of the average user's everyday activity. By integrating them in a personal agent, we can gather relevant information about the user and improve the user's experience. Each social network profile can be linked to several identities. As we saw before, this means the events from different profiles will share a common namespace, making it easy to subscribe to all of them.

The *Location service* makes it possible to set locations to each identity. Events are sent every time there is a location change, or when a module queries the location of an identity. The *Calendar service* is a common interface to deal with calendars from different sources. It is especially meant as an abstraction for online calendar services. The *Information service* offers a simple unified interface for agents to query information from external information sources. As of this writing, the Information service supports SPARQL, being able to send queries to multiple endpoints (DBpedia, data.gov, etc.). The *Transaction service* makes it easier for agents to handle operations with online services that follow a known pattern. For instance, the processes between booking a flight and arriving safe to the destination accommodation are quite similar regardless of the flight company, shuttle bus operator, etc. Given that, the Transaction service identifies different events as steps in such processes and acts accordingly to offer extra information to the agents. The *Semantic reasoner* is a semantic inference engine offered to be used on demand. This is a service offered primarily to the Personal Agent, to relieve it of including a semantic engine. Additionally, although the Rule Engine of the TAS is also a semantic engine so it does not require using the semantic reasoner, it may be configured to be used to reduce the load of the Rule Engine, precomputing some of the inferences and inject them as new events. The *LOD instance linker* is in charge of querying the semantic endpoint it is connected to (e.g. FreeBase, DBPedia, LOV) and extract additional information from the semantic instances received. The LOD is in charge of executing the instance and property linkage included in the vocabularies described in Section 4.3.4. It makes use of the semantic reasoner to perform this task.

5.3.4 Communication between TAS clients, TAS server and personal agents

In this section, we describe how different components communicate, taking advantage of the modularity of the architecture. In particular, the architecture is required to orchestrate the event distribution among i) the TAS server, ii) the TAS clients, iii) the distributed sensor network and Web services, and iv) the Personal Agent.

The Event Manager is the main actor in the communication process, and it is directly connected to the TAS's execution planner, which from Event Manager's perspective may be seen as one of the plugins (as described in Section 5.3.3). This direct communication transfers to the TAS all events coming from the Evented Web Bus, and receives the events it generates. Instead of using an adapter to the Evented Web Bus, the direct communication was preferred to reduce the load of the event subscription handler and improve performance. This is not the case of the execution planner from the mobile client. In this case, the mobile client has direct access to Web services and sensor networks using its own adapters. Thus, there is no need to keep an open communication between the event manager and the execution planner. Therefore, the messages exchanged to orchestrate the execution of the automations with a Mixed Execution Profile will be driven trough the Agent Bus. One of the interesting features this architecture offers to mobile clients, is that storing the message when the client is disconnected, and sending them all when it is back available. But this feature is only available when the access to the Web Service is done using the buses instead of direct connectors from the client.

The Web Client and the Mobile Client, communicate with the agent and the TAS using the Agent Bus (also known as the Client Bus). In addition to supporting the Mixed Execution Profile as just explained, this is used to inform the TAS of i) new channel activations, ii) creation, modification and deletion of automation rules, and iii) disconnection periods. Channel activations are performed in the client side, once activated the information is sent to the TAS to update the channel directory. The information about creation, modification and deletion, when received is stored in the Rule Repository. This may be a bidirectional operation to handle synchronisation of multiple clients with local rule repositories. The client also informs the architecture of disconnection issues, to that the event manager can forward all stored messaged the client was subscribed to. To receive information from the TAS, the clients have to subscribe the to the messages they want to receive.

The Personal Agent communication with the TAS is performed throughout the Agent Bus too. As with the client, the Personal Agent has to subscribe to the messages it is interested in.

5.4 Personal agent plan library

As seen in Section 5.2.1, Personal assistants help users in their day-to-day activities, releasing them from performing repetitive time consuming tasks. This functionality overlaps with task automation of TASs. For instance, tasks developed by information management agents in charge of organising email attachments may be replaced by a set of automations similar to "when I receive an email from my boss, save the attachment to the Dropbox folder in named 'from-boss' ". With reminder agents, it is even more obvious that they may be replaced using rules like "Open a toast notification in my smartphone five minutes before the start of my meetings".

However, strictly speaking, personal agents usually overtake TASs, either because their plans are more flexible and have the ability to take decisions reasoning on the context information, or simply because they are proactive and the user does not need to instruct them to act. Nonetheless, the personal agent designed in this chapter is not intended to perform tasks that may be accomplished using automations from a TAS but to assist the user while using TAS in their daily life.

5.4.1 Types of behaviour

The plan library of the personal agent for TASs is oriented to assist the users in creating, updating, and managing the automations rules of their portfolio. Below, we present the list of proactive behaviours the personal agent for TAS may perform to achieve these goals.

Recommend popular automation rules. One of the most obvious proactive behaviours of the personal agent for TAS is suggesting popular automations to the users. To enable this behaviour the agent is granted access to the automation directory of the TAS (described in Section 4.3.1) where the information about automations is stored, and also to the Channel Directory from the TAS architecture (Section 2.3) that stored the list of active channels available for is user. Thus, knowing which are the active channels for the user, the agent may filter those automations that are feasible to the users and suggest them, i.e. those automations where the channels involved are active for the user.

Suggest automation rules based on recently activated channels. This is a behaviour similar to the former one, but it is triggered by recently activated channels.

Learn rules from user behaviour. Personal agents have access to context information, since they receive all incoming events from all user channels. This a powerful source of information that may be used to learn the routines of the users, and propose them to automate those routines. For instance, the personal agent may detect that every morning, a few moments after a user arrives to work she mutes her smartphone. The agent have access to the event stream of the user, so it inspects which events happened a few moments prior to muting the phone –in this case, the positioning service announced she arrived at the office. When a sequence event-action is repeated several times, the agent suggests the user to automate that routine.

Update rules based on changes in the context. Personal agents also have access to Channel Directory from the TAS architecture (Section 2.3) where the information about the service of channels is stored. It registers when channels are deactivated, temporary unavailable, or out of coverage range. When a channel is no longer available and it is used in one or more automations, the agent searches for an alternative among the active channels. If any, a modification in the rule is suggested until the service of the initial channel is restored. For instance, if a user decides to migrate to Bitbucket, and closes her Github account. The Github channel becomes inactive for the user and all the automations are halt. The agent gets aware of the situation, taking advantage of the semantic description of the channels –using the specific vocabularies generated in Chapter 4 (Section 4.3.4)– it finds the Bitbucket channel as a replacement which is suggested. Then, all the Github related automations are migrated to use the Bitbucket channel.

Duplicate rules in similar environments. In the particular case of device channels (Section 2.2.2), being under coverage is particularly important. When a user gets away from a sensor, the channel is marked to be out of coverage, thus these device channels need to announce themselves –according to the discovery paradigm– to be activated when back into coverage. This fact may be used by the agent to adapt existing rules to different environments. This is one of the behaviour described in Sarah's scenario in Chapter 2 (Section 2.1). Consider the user (Sarah) has her home's SmartTV channel activated, and she orchestrated an automation rule to lower the TV volume when she has an incoming call. Taking advantage of the semantic description of the channels, the agent discovers that when Sarah is at her friends house the SmartTV is compatible with that automation, and it suggests adapting the automation to this environment.

Alert of rules with low usage. Personal agents track the execution of the automation. Therefore, it has information about the frequency automations are executed, and also about the last execution time. When it has been a long time since the last time an automation was executed, the agent informs the user and ask permission to remove it, with the aim of keeping the automation portfolio clean.

Deactivate inconsistent rules. One of the side effects of not keeping the automation profile clean, is that as it becomes bigger, the changes of having collisions in the rules increases. This situation was described in Section 2.2.3. The simplest situation where two rules may collide, is when two automations triggered by the same type of event try to execute incompatible actions. Using the semantic description of the actions, the agent is able to detect incompatible actions and warn the user about these situations.

5.4.2 Plan implementation examples

Although the chapter is not intended to be code intensive, we considered it is convenient to illustrate some of the behaviours just described. Thus, we present below a few scripts with a tentative implementation of some agent behaviours. Each script presents a list of agent plans in AgentSpeak (Bordini and Hübner, 2005), and it is commented for easier understanding. The implementation of the plans assumes the environment of the agent system is connected to the TAS database, and injects in the agent belief base information about the user's events and actions, as well as the channels.

The script shown in Listing 5.1 presents a set of plans to propose the user to create automations taken from the list popular automations for recently activated channels. It matches the behaviour *suggest rules based on recently activated channels* described in the former section.

Listing 5.1: Plans to learn automations from user's routine.

```
// Every time a new channel is active,
// for every popular_automation in the automation directory that matches
// the channel, if the second channel involved is also active
// suggest the user to user the automation
+channel(ChannelId, active)[source=user] :
    popular_automation(automation(ChannelId, OtherChannel), Relevance) &
    Relevance > 0.5 &
    channel(OtherChannel, active)
    <-
    !suggest(create, Automation) .

// Handle suggestions asking the user when available
+!suggest(create, Automation) : user_available
    <-
    .ask_user("Do you want to create this automation?");
    tas.describe(Automation) .

+!suggest(create, Automation) : not user_available
    <-
    .at("now +1 h", "+!suggest(create, Automation)") .

// Handle response
+response(create, Automation, true)
    <-
    tas.create_automation(Automation) .

+response(create, Automation, false) <- True
```

The script shown in Listing 5.2 presents a set of plans to propose the user to create automations based on the routines learnt from the user behaviour.

Listing 5.2: Plans to learn automations from user's routine.

```
// Analyse event registry every time an action is taken.
// the internal function add to the KB routine facts
+action(Id, Action_payload) <-
    context.analyse_recent_events(Id) .

// When a routine repeats more than five times suggest to automate it
+routine(Event, Action, Times) : Times > 5
    <-
    !suggest(create, automation(Event, Action)) .

// Handle suggestions asking the user when available
+!suggest(create, Automation) : user_available
    <-
    .print("Do you want to create this automation?");
    tas.describe(Automation) .

+!suggest(create, Automation) : not user_available
    <-
    .at("now +1 h", "+!suggest(Automation)") .
```

```
// Handle response
+response(create, Automation, true)
    <-
    tas.create_automation(Automation) .

+response(create, Automation, false)
    <-
    ?automation(Automation, Event, Action) ;
    -routine(Event, Action) .
```

The script from Listing 5.3 shows a set of plans to analyse the automation profile of the users and detect unused rules. Those are marked to be removed after user's approval.

Listing 5.3: Plans to remove unused automations.

```
// Everyday analyse the automations and check their last exection time
+!analyse_rules(User)
    <-
    .findAll(automation, User, LA) ;
  !analyse_rule(User, LA) ;
  .at("now + 24 h", "+!analyse_rules(User)") .

-!analyse_rules(User) <- .at("now + 24 h", "+!analyse_rules(User)").

// Analyse last time a rule was used.
// If it is more than three let the user know
+!analyse_rule(User, automation(Aut, lastUsed(D, M, Y))|LA)  :
    .date(DD,MM,YY) &
    std.compare_date(date(D,M,Y), date(DD,MM,YY), Days) &
    Days > 90
    <-
    !suggest(remove, Aut) ;
    !analyse_rule(User, LA) .

// otherwise
+!analyse_rule(User, automation(_, _|LA) <- !analyse_rule(User, LA).

// Handle suggestions asking the user when available
+!suggest(remove, Automation) : user_available
    <-
    .print("This automation has not been usd for 3 months");
    .print("Do you want to remove it?");
    tas.describe(Automation) .
```

```
+!suggest(remove, Automation) : not user_available
    <- True . // Make the suggestion other day

// Handle response
+response(remove, Automation, true)
    <-
    tas.remove_automation(Automation) .
```

These scripts present a few examples of the implementation of the behaviours above. They are shown for illustrative purposes, thus they do not contain all plans that manage all the cases in order to assure the goal achievement, and for sake of readability and expressibility they are not fully compliant with AgentSpeak syntax. In addition, they rely on a few functions that implement communication with the user, and the creation and deletion of automation, and also on the injection of beliefs into the BB that are received from the bus, as a consequence a user action, or other contextual event.

5.5 Case study: Managing a smart environment

This case study presents a personal agent for managing and customising smart workspaces. It aims to assist users in creating automation rules that involve using sensors and actuators from the connected environments.

Each connected environment is organised in hot-spots. They consist on a grid of connected beacons that provide information about the location of the users inside the workspace; a few connected light switches that allow the system to switch on/off the plugged lights or devices, and send event messages when they are manually operated; and a connected door lock –installed at the entrance door– to be aware when the door opens, that also lets the system open the door remotely by connecting to the door lock. In addition, a few Web services are involved (Twitter and Google Calendar) as well as some smartphone resources (NFC, messaging).

The particularisation of the TAS architecture (Figure 5.1) for this implementation is shown in Figure 5.2. The TAS connects to the beacons and other connected devices using a set of adapters. In particular, the beacons offer two modes of functioning: they send a notification to a hub server every time a user is located, and they also communicate with the smartphone whose NFC tag was identified. In this case study, the first mode is used since it is easier to gather the information from the beacon hub. Similarly, the door lock and the switches are connected to the Internet, they send notifications to the server every time

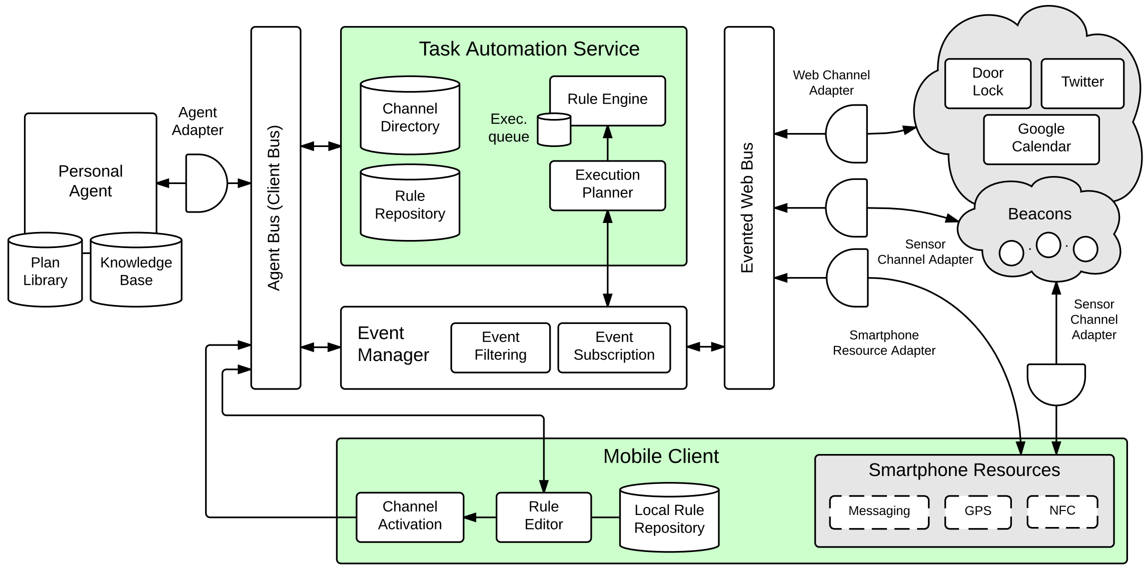


Figure 5.2: Architecture of the event-based task automation prototype.

an event happens (e.g. manually switching the light, opening the door, etcetera). A mobile client is included, not only to be used as a NFC tag to let the beacons know the user location, but also to provide the user a rule editor and show the communication capabilities of the Agent Bus to connect the Mobile client and the central TAS. The activation of the channels (Google calendar, Twitter) is also handled in the mobile client. Finally, the personal agent is connected to the TAS using the MAIA Agent Bus as usual. In this scenario, the personal agent is subscribed to all incoming messages to be able to assist the user in automating the environment.

We model the channels using the specific vocabularies for *connected-spaces* learnt in Chapter 4. An excerpt of the beacons channel is shown in Listing 5.4. It includes mappings to event properties from the SSN ontology for describing observations, and from DBpedia, for unifying the property used to provide a distance.

Listing 5.4: Beacons channel excerpt (channel and events).

```
# Channel definition
ewe-presence:PresenceSensor a owl:Class ;
    rdfs:label "Connected presence sensor"@en ;
    rdfs:comment "This channel represents a presence sensor."@en ;
    rdfs:subClassOf ssn:SensingDevice ;
    rdfs:subClassOf ewe:Channel .

# Events definition
ewe-presence:PresenceDetected a owl:Class ;
```

```
    rdfs:label "Presence Detected"@en ;
    rdfs:comment "This Trigger fires every time your presence sensor detects
        presence at any distance."@en ;
    rdfs:subclassOf ewe:Event ;
    ewe:generatedBy ewe-presence:PresenceSensor ;
    rdfs:subclassOf ssn:ObservationValue .

# No actions defined for this channel

# Parameter mappings
# Use observe property from ssn
{ ?event ewe:generatedBy ?channel .
  ?channel a ewe-presence:PresenceSensor . }
=>
{ ?channel ssn:observes ?event . }.

# Mapping for DBpedia distance
{ ?event ewe:generatedBy [a ewe-presence:PresenceSensor] .
  ?event ewe:distance ?distance . }
=>
{ ?event dbpedia-owl:distance ?distance . }.
```

In this scenario, the central TAS executes the automations using the EYE reasoner [1], which is compatible with N3 rules. EYE rules constitutes an alternative to SPIN rules shown in Chapter 3, and show the flexibility of the EWE ontology in this matter. The system has been tested, loading several automation rules using the rule editor in the mobile client. The automation rules are created, loaded into the user profile, and executed when the triggering event happens. The automation rules may use any of the channels previously described. On the other hand, the agent system, which is subscribed to all events, receives all information that sent to the TAS and stores it in its belief base, to be used in plans and goal achievement

In this scenario, the automations that are related to a hot-spot, are stored in the rule database in relation to the hotspot they are related to. This way, these automations are only active in the context of the hotspot, and letting the user to load or unload all the automation at once.

This use case is intended to show i) that the proposed architecture provides access to data-streams of different nature; ii) that the agent system has access to those data-streams too; iii) that the automation rules, created in the mobile client, are sent to the TAS and stored there; and iv) that the automation rules are executed in the rule engine. Furthermore, it is also intended to show functionalities that are beyond the implementation. This is the

[1]http://n3.restdesc.org/

case of the scenario described below.

Within this environment, a user may create automations that read “When I get to work, tweet a random tweet from a morning-tweets list”, or “text me when someone sits at my desk”. However, the aim of this case study is to show how the personal agent can learn from users routine and propose automations. Consider a daily-routine of a common user named Anna. In the morning, when Anna gets to the connected workspace the beacon situated at the entrance detects she just arrived. She uses the ID card to unlock the door, gets to her desk, and switches on the computer. The beacon at her desk detects she is sitting there. After checking the email, she goes to the coffee machine (where another beacon is situated) and has cappuccino. Meanwhile, her co-workers have been arriving. Around 11:00 am one of the co-workers goes to the coffee machine and immediately some others, Anna included, join him to have a coffee together.

At this point, the reader can identify several routines that repeats every day, and so does a personal agent. These routines are subject to be automated. For instance, after a few days, the personal agent may suggest Anna to create an automation rule to automatically open the door when the beacon at the entrance detects her. Anna considers this automation very convenient, since she no longer needs to get the ID out of her purse. The implementation of the plans of the personal agent for learning automations from routines are shown in Listing 5.5[2].

Listing 5.5: Plans to learn automations from user’s routine.

```
// Analyse event registry every time an action is taken.
// the internal function add to the KB routine facts
+action(Id, Action_payload) <-
    context.analyse_recent_events(Id) .

// When a routine repeats more than five times suggest to automate it
+routine(Event, Action, Times) : Times > 5
    <-
    !suggest(create, automation(Event, Action)) .

// Handle suggestions asking the user when available
+!suggest(create, Automation) : user_available
    <-
    .print(``Do you want to create this automation?'') ;
    tas.describe(Automation) .

+!suggest(create, Automation) : not user_available
```

[2]This is taken from plan implementations described in Section 5.4

```
        <-
        .at("now +1 h", "+!suggest(Automation)") .

    // Handle response
    +response(create, Automation, true)
        <-
        tas.create_automation(Automation) .

    +response(create, Automation, false)
        <-
        ?automation(Automation, Event, Action) ;
        -routine(Event, Action) .
```

Moreover, this scenario also supports multi-user rules as defined in Section 2.2.3. For instance, the agent may detect that a few moments after user is near the coffee machine and it is around 11:00 am, other users start going there too. Thus, a suitable automation could be to send a text message to all co-workers, as soon as one user is at the coffee machine, to let them know it is coffee time. This situation is much more complex that the former routines, and it involves additional reasoning to decide sending the messages; however, with the necessary adjustments, the agent could address this automation too. Similarly, the lights or the security alarm may be automated taking into account if there is any user at the working space.

The approach presented in this use case shows an interesting situation, where the personal agent no longer needs to include plans to execute the task automations as personal agents in the state of the art do. In this case, the intelligence of the agent is at a higher level, learning from the context which automations should be created. In addition, this also has advantages to the user who is in total control of the automations because she can modify or even delete them using the rule editor of the TAS.

5.6 Discussion and outlook

The architecture presented in this chapter proves that it is possible to achieve modern systems that combine the potential of intelligent agent systems and the interconnection and ever-growing applications of the modern web.

The resulting application goes beyond the state of the art, putting together already existing solutions from different fields. It thus shows that we can make good use of the existing technologies to implement innovative ideas. It is important to note that the most important shift is in the way we understand agents and agent communication. Adapting existing systems and frameworks to MAIA also requires work, especially in the case of Multi Agent Systems. However, such adaptation only needs to be done once, and it allows its connection to a wide range of modules.

The Modular Architecture for Intelligent Agents has been integrated within the Reference Architecture of TAS to give the agent systems access to the features of TASs. The architecture has been validated in two case studies, in the first one as a independent architecture i.e. without a TAS, and in the second case study in combination with a TAS. It has been validated that the modularity and loose couple design favours the integration of personal agent in TASs.

In this chapter, the different behaviours that a personal agent may adopt to assist the user while using a TAS has been studied. As a result, seven different proactive behaviour have been documented, as well as another supporting reactive behaviour. Moreover, some of these have been implemented using AgentSpeak, and used in the case studies.

There are several aspects in which MAIA can be extended or improved. It also opens the discussion about the integration of the evented programming paradigm and the design of BDI agents.

One of the main aspects to improve from a pragmatic point of view is the security of the information being exchanged and the scope in which it is visible. Currently MAIA allows username/password authentication and mechanisms to control event subscription on a per-module basis.

Another field for future research is to further expand the definition of events to include other concepts such as propagation of events. This might lead to delegation and collective planning, but it also poses challenges related to agent communication.

CHAPTER 6

Conclusions and Future Work

This chapter summarises the conclusions of the thesis. It presents the most relevant contributions obtained from this research journey, and aims to condensed it as a sequence of decisions, motivated by insights, that drove the study —ranging from the motivational situation to the final conclusions. Additionally, the research done has enabled to identify potential research lines which are also discussed here.

6.1 Outlook

Task Automation Systems (TAS) have experimented a remarkable increase of its popularity, which is reflected in their number of users. To have a broad view of the scene, in the second chapter we presented a thorough review of the most relevant TASs, since the task automation market is already very wide and embraces players of very different nature. Nonetheless, new players are continuously entering the stage, and not only new web-based TASs, but also TASs that execute as mobile apps, TASs that are embedded in smart home hubs, or even connected devices that integrates as channels in third party TASs. Crowdfunding[1] platforms such as Kickstarter[2] or Indiegogo[3] are a breeding ground for new TASs and TAS components. Monitoring the projects on these platforms may give an idea of the movements in the task automation market. For instance, considering only Kickstarter, in the last two quarters of 2015, there appeared 14 projects developing products that fit the TAS approach, and 67 more products that are likely to be integrated as TAS channels. This is more than two new TASs per month, and more than six new channels per month, on average, in a single crowdfunding platform. A second evidence of growth is the channel and functionality expansion made by existing TASs. As an example, Ifttt (Ifttt, 2015) increased the number of channels integrated in its platform from 97 to 238 between the first quarter of 2013 and the last quarter of 2015 —this is an average of 3.91 new channels each month. Meanwhile, Zapier (Zapier, 2015) took its channel offer much further, increasing the number of supported channels up to 621 by the end of 2015.

In this novel and growing scenario we identified several challenges that worth be addressed and solved. First, the concept of Task Automation Services is still fuzzy. Some researchers consider they borrow inspiration from mash-up technology (Vladimir et al., 2015), so it is legitimate to uphold that they may be considered as a deviation of this technology. After analysing their similarities and differences, we concluded TAS and mash-ups have substantial differences so there is room for a formal definition of TAS, as we argued in the second chapter. Second, the current evolution of TASs –taking in new players and expanding the channel offer, tends to the democratisation of the technology, making it more accessible to users and developers. As a consequence, it will also increase the capabilities and features of future TASs systems, but there is still a technological gap that should be filled. We identified that a common model for describing channels and automations provides several advantages such as supporting interconnection between TAS, facilitating the

[1]Crowdfunding is the practice of funding a project or venture by raising monetary contributions from numerous people, today often performed via Internet-mediated registries

[2]http://www.kickstarter.com

[3]http://www.indiegogo.com

exposure of services and connected devices as channels, enabling automations with reasoning over external services, etc. Therefore we proposed the EWE ontology, described in the third and fourth chapters. Finally, the evolution of other similar services and platforms that are also centred on the user is to become more proactive, offering the service or the information to the user even before it asks for it. For instance, Google Now[4], Microsoft's Cortana[5] or Apple's Siri[6] are prominent examples of this trend. In this regard, TAS have a lot of potential, e.g. users may be assisted when creating automations and recommended with useful automations based on their profile. We proposed an agent architecture for TAS that supports common task automation features with the proactivity given by the agent.

6.2 Conclusions

Throughout the course of the thesis, a number of contributions have been delivered that can be gathered under three main contribution areas:

A reference model for describing TAS. This thesis proposes a reference model for describing TAS, which has been build upon the insight gathered from comparing the features of twenty-one TASs. To formally extract the features of these TAS, a comparison framework was proposed. It is based on 18 criteria developed around 6 dimensions of the three main components of TAS: channels, rules and the TAS platform. As result of the formalisation of the reference model, this thesis coined the term Task Automation Service (TAS). The model has been implemented as an ontology named Evented WEb ontology (EWE). Apart from the advantages of reference models, EWE introduces five additional advantages i) it enables exporting the automation portfolio, ii) it enables reasoning over external resources, iii) it allows external software agents to read, understand ant thus reason with the data, iv) it facilitates the entrance of new actors in the market, and v) it also facilitates the analysis of domain data. Finally, the chapter exhaustively evaluates the EWE ontology proposed by using a data-driven approach. The evaluation conclusion is that EWE effectively models four popular commercial TASs while it addresses the shortcomings observed in them.

Ontology learning methodology for extracting TAS vocabularies. The thesis proposes a methodology to automatic ontology learning about specific channel domains. The process consist of four steps that have been thoroughly validated. First, using

[4]https://www.google.com/landing/now/
[5]http://windows.microsoft.com/en-us/windows-10/getstarted-what-is-cortana
[6]http://www.apple.com/ios/siri/

the channel and automation directory of TAS a dataset of channels has been compiled. This is relevant information to be used by software agents and users to consult information about the TASs, and it has been used by the implementation of the personal agent, to suggest automations in various ways. Then, a measure of similarity that combines semantic structure and linguistic similarity has been proposed. It is presented as a correction of the Jaccard similarity, and it has been evaluated in the former dataset concluding the results of the corrected similarity are more accurate, in the domain under study, than those obtained using the original similarity. Next, a clustering algorithm was designed to group channel in small clusters which expose shared functionalities. It has been proved that this clustering algorithm presents advantages in this scenario such as, i) it does not require to number of cluster a priori, ii) individuals that do not match any cluster are left unclustered, iii) individuals may belong to more than one cluster, and iv) each cluster provides the features that support that cluster. Finally, we used the harvested instances, the similarity metric and the clustering algorithm to automatically learn ontologies. The results of the whole process have been evaluated at a dataset level, and at a ontology level obtaining a combined F-score of 86.82%.

Personal agent architecture for task automation. The thesis proposes a Modular architecture for Intelligent Agents (MAIA) to providing agent platforms an easy and transparent communication channel to Web services, sensor networks, user interfaces, etcetera. Although it has been designed as a stand-alone architecture, it has been integrated within the Reference Architecture of TAS to give the agent systems access to the features of TASs. The architecture has been validated in two case studies, in the first one as an independent architecture i.e. without a TAS, and in the second case study in combination with a TAS. It has been validated that the modularity and loose couple design favours the integration of personal agent in TASs. Moreover, the thesis studies the different behaviours a personal agent may adopt to assist the user while using a TAS, and those have been implemented in the case studies.

6.3 Future research

The development of this thesis and its contributions to the state of the art in Task Automation have opened new possibilities for future research. The case scenarios and evaluation processes described have proof the usefulness of the contributions, but at the same time they have unveiled different approaches that, although not being addressed in this work, worth being mention here. In terms of conclusions for the thesis research, the following lines of future research can be pointed out as follows.

The survey about commercial Task Automation Services described in Chapter 2, not only set the foundations of the research conducted along this thesis, but also discover many interesting features and possibilities of TASs that have not been addressed in depth. This is the case of group channels and group rules, a concept still not covered by the state of the art, neither by commercial TASs. A sample of their potential is shown in Section 5.5, where a few automations considering events coming from data-streams that belong to different users are explained. The future work in this matter covers a proper definition of group channels, including their usage restrictions between users. Different usage policies may be considered: first connected user has priority, fixed priorities, shared priority, etcetera. Once group channels are defined, group rules may be addressed. These are automation rules that may be triggered by event related to different users or a combination of those. For instance, at a working space, a rule for switching on/off the heating system depending on who is in the room could be considered. However, group rules are strongly associated to the concept of collisions. In the former example, a collision would be caused by two different users setting the temperature to different values when they are at the room. Collision has already been addressed in this thesis, considering only the simplest cases where two rules triggered by the same event try to execute incompatible actions, i.e. actions that put a channel into incompatible estates. However, the ontology does not cover the definition of these restrictions, thus it does not provide support to identify collisions. Moreover, the concept of collisions in groups channels is even more challenging, because the conflict may be caused by rules of different users and in that case a resolution process must be considered. The simplest resolution process consists on using priorities, but different process that consider lower impact may be more appropriate. This concept refers to a mechanism to resolve collisions without taking into account the usage priority, and instead infer the deactivation of which automation rule may cause fewer harm.

In Chapter 2 the concept of Mixed Execution Profile was introduced. Although some of its features and requirements are introduced and even addressed in the Reference Architecture, the execution of automation rules with a Mixed Execution Profile requires the definition of coordination algorithms to orchestrate the exchange of command between the central TAS and the mobile TAS, or more precisely between the execution planners of both TAS. A first approach to this concept was presented by Coronado et al. (2014a). Moreover, the concept of Mixed Execution Profile offers several advantages to the industry that could drastically reduce the bandwidth usage, or allow users to have automations event in areas without Internet coverage.

The EWE ontology presented in Chapter 3, effectively describes TASs. For design decisions, features related to rule execution, the channel connection (either device of web service), authorisation were left apart. However, the concept of group channels and rules, and collisions require the support of an ontology to be efficiently handled. Thus, an extension of EWE's core could be considered to address all issues mentioned. Similarly, the orchestration of the mixed execution profiles may benefit from an EWE's extension too.

From the point of view of the final user, the semantic definition of TAS channels allow them to search for channels and automations across different TAS. To allow the user perform these queries, the semantic endpoint of channels presented in Chapter 3, as a proof of concept should be deployed and provided with a user interface. In combination with the automatic scraping of the channel and automation instances, the information contained in the endpoint may be always up to date. This also enables further research in semantic introspection and linking of channels instances to those from the LOD cloud.

In Chapter 4, the EWE ontology was complemented with the specific vocabularies learnt. However, analysing the data from the harvested automations was out of the scope of this thesis. Having harvested more than 350k automations, the dataset constitutes a valuable resource for exploring user profiles from the point of view of automations, analysing substitutive channels, etcetera. This information may also be used to train a recommender system of automations based on user profiles. This is also an interesting line of future research.

In Chapter 5 a distributed and modular agent architecture was presented. It was used to easily communicate TAS with agent systems as well as with connected devices and web services. Several use cases were implemented to proof its feasibility, and they were used to present the functionalities personal agent introduces to task automation services. However, it was out of the scope of this thesis providing a formal evaluation of these systems from the point of view of Human Computer Interaction. It is an interesting future research line that could validate the usefulness of personal assistants in TASs environments. Performing this evaluation requires the development and deployment of a stable version of semantic TAS which provides access to a sufficient number of channels so that the users use the system on a daily basis.

Moreover, this system enables the evaluation of the vocabularies learnt in Chapter 4 using an application based approach (Hazman et al., 2011) which would complement the current approach.

Another interesting addition to the user interface of the personal assistant for TAS consist on including a NLI interface. This has already been tackled by Van Kleek et al. (2010) providing a constrained NLI, and also in other application fields (Coronado et al., 2015), or commercial like Apple's Siri or Microsoft's Cortana.

Bibliography

RIF Production Rule Dialect. `http://www.w3.org/2005/rules/wiki/PRD`, 2008. Accessed: 2015-06-10.

RuleML Overview and Motivation. `http://wiki.ruleml.org/index.php/RuleML_Home`, 2010. Accessed: 2015-06-10.

SPIN Overview and Motivation. `http://www.w3.org/Submission/spin-overview/`, 2011. Accessed: 2015-06-10.

K. Aberer, M. Hauswirth, and A. Salehi. Infrastructure for Data Processing in Large-Scale Interconnected Sensor Networks. In *2007 International Conference on Mobile Data Management*, number 1, pages 198–205. Conference and Custom Publishing, May 2007.

B. Alfonso, E. Vivancos, V. Botti, and A. García-Fornes. Integrating jason in a multi-agent platform with support for interaction protocols. In *Proceedings of the compilation of the co-located workshops on DSM'11, TMC'11, AGERE!'11, AOOPES'11, NEAT'11 and VMIL'11*, SPLASH '11 Workshops, pages 221–226, New York, NY, USA, 2011. ACM.

B. Amini, R. Ibrahim, M. S. Othman, and A. Selamat. Capturing scholar's knowledge from heterogeneous resources for profiling in recommender systems. *Expert Systems with Applications*, 41(17):7945 – 7957, 2014.

B. Amini, R. Ibrahim, M. S. Othman, and M. A. Nematbakhsh. A reference ontology for profiling scholar's background knowledge in recommender systems. *Expert Systems with Applications*, 42(2):913 – 928, 2015.

Atooma. Atooma a touch of magic. `http://www.atooma.com/`, 2015. Accessed: 2015-03-31.

J. C. Augusto and C. D. Nugent. The Use of Temporal Reasoning and Management of Complex Events in Smart Homes. In *Proceedings of the 16th European Conference on Artificial Intelligence*, Valencia, 2004.

Automateit. Automateit turn your smartphone into a genius-phone. `http://automateitapp.com/`, 2015. Accessed: 2015-03-31.

J. Bae, H. Bae, S.-h. Kang, and Y. Kim. Automatic control of workflow processes using ECA rules. *IEEE Transactions on Knowledge and Data Engineering*, 16(8):1010–1023, August 2004.

D. F. Barbieri, D. Braga, S. Ceri, E. Della Valle, and M. Grossniklaus. C-SPARQL: SPARQL for continuous querying. In *Proceedings of the 18th international conference on World wide web*, pages 1061–1062. ACM, 2009.

F. Barbon, P. Traverso, M. Pistore, and M. Trainotti. Run-Time Monitoring of Instances and Classes of Web Service Compositions. In *International Conference on Web Services, 2006*, 2006.

A. Beach, M. Gartrell, X. Xing, R. Han, Q. Lv, S. Mishra, and K. Seada. Fusing mobile, sensor, and social data to fully enable context-aware computing. In *Proceedings of the Eleventh Workshop on Mobile Computing Systems & Applications*, page 60, New York, February 2010. ACM Press.

S. Bechhofer. OWL Reasoning Examples. `http://owl.man.ac.uk/2003/why/latest/`, 2003. Version 03/12/2003.

W. Beer, V. Christian, A. Ferscha, and L. Mehrmann. Modeling Context-Aware Behavior by Interpreted ECA RulesIn , *Euro-Par 2003 Parallel Processing*, volume 2790 of *Lecture Notes in Computer Science*, pages 1064–1073. Springer Berlin Heidelberg, 2003.

F. L. Bellifemine, G. Caire, and D. Greenwood. *Developing Multi-Agent Systems with JADE (Wiley Series in Agent Technology)*. John Wiley & Sons, 2007.

T. Berners-Lee. Notation 3 Logic. `https://www.w3.org/DesignIssues/Notation3`, 2011. Accessed: 2016-01-10.

T. Berners-Lee and D. Connolly. Notation3 (N3): A readable RDF syntax. `http://www.w3.org/TeamSubmission/n3/`, 2011. Version 28/03/2011.

T. Berners-Lee, J. Hendler, O. Lassila, et al. The semantic web. *Scientific american*, 284 (5):28–37, 2001.

D. Berrueta, D. Brickley, S. Decker, S. Fernandez, C. Gorn, A. Harth, T. Heath, K. Idehen, K. Kjernsmo, A. Miles, A. Passant, A. Polleres, and L. Polo. SIOC Core Ontology Specification. `http://rdfs.org/sioc/spec/`, 2010. Version 25/03/2010.

H. Boley and M. Kifer. Rule Interchange on the Web. *New York*, (September):269–309, 2007.

H. Boley, A. Paschke, and M. O. Shafiq. RuleML 1.0: The Overarching Specification of Web RulesIn , *Semantic Web Rules - International Symposium, RuleML 2010, Washington, DC, USA, October 21-23, 2010. Proceedings*, volume 6403 of *Lecture Notes in Computer Science*, pages 162–178. Springer, 2010.

A. Bolles, M. Grawunder, and J. Jacobi. *Streaming SPARQL-extending SPARQL to process data streams*. Springer, 2008.

R. H. Bordini and J. F. Hübner. Bdi agent programming in agentspeak using jason. In *Proceedings of 6th International Workshop on Computational Logic in Multi-Agent Systems. Volume 3900 of lncs*, pages 143–164. Springer, 2005.

J. Brank, M. Grobelnik, and D. Mladenić. A survey of ontology evaluation techniques. In *Proceedings of 7th International Multi-conference on Information Society*, Ljubljana, 2005.

I. Bratman. Plans, and practical reason. *Cambridge, Mass.: Harvard UP*, 1987.

J. G. Breslin, A. Harth, U. Bojars, and S. Decker. Towards semantically-interlinked online communities. In *The Semantic Web: Research and Applications*, pages 500–514. Springer, 2005.

D. Brickley and L. Miller. FOAF Vocabulary Specification 0.99. `http://xmlns.com/foaf/spec/`, 2014. Version 14/01/2014.

P. Buitelaar, P. Cimiano, and B. Magnini. *Ontology learning from text: methods, evaluation and applications*, volume 123. IOS press, 2005.

V. Bullard, B. Murray, and K. Wilson. *An Introduction to WSDM*. OASIS, 2006.

P. Cimiano and J. Völker. Text2onto - a framework for ontology learning and data-driven change discovery In , *Proceedings of the 10th International Conference on Applications of Natural Language to Information Systems (NLDB)*, volume 3513, pages 227–238, Alicante, Spain, Juni 2005. Springer.

P. Cimiano, A. Mädche, S. Staab, and J. Völker. Ontology learning. In *Handbook on ontologies*, pages 245–267. Springer, 2009.

Cloudwork. Cloudwork connect your business apps. `http://cloudwork.com/`, 2015. Accessed: 2015-03-31.

M. Coronado and C. A. Iglesias. Task Automation Services: Automation for the masses. *IEEE Internet Computing*, 99:52–58, 2015a.

M. Coronado and C. A. Iglesias. EWE ontology specification. http://www.gsi.dit.upm.es/ontologies/ewe/, 2015b. Accessed: 2015-03-31.

M. Coronado and C. A. Iglesias. Task automation services study. http://www.gsi.dit.upm.es/ontologies/ewe/study/full-results.html, 2015c. Accessed: 2015-03-31.

M. Coronado, R. Bruns, J. Dunkel, and S. Stipković. Context-awareness in Task Automation Services by Distributed Event ProcessingIn , *Proceedings of the 15th Internacional Conference of Web Information System Engineering (WISE 2014)*, number 191-202, 2014a.

M. Coronado, R. Bruns, J. Dunkel, and S. Stipković. Context-awareness in Task Automation Services by Distributed Event ProcessingIn , *Proceedings of the 15th Internacional Conference of Web Information System Engineering (WISE 2014)*, number 191-202, 2014b.

M. Coronado, C. A. Iglesias, and A. M. Mardomingo. A Personal Agents Hybrid Architecture for Question Answering featuring Social Dialog. In *2015 International Symposium on INnovations in Intelligent SysTems and Applications*, September 2015.

A. Costa, J. C. Castillo, P. Novais, A. Fernandez-Caballero, and R. Simoes. Sensor-driven agenda for intelligent home care of the elderly. *Expert Systems with Applications*, 39(15): 12192 – 12204, 2012.

C. Crespo, M. Coronado, and C. A. Iglesias. DrEWE an intelligent platform for task automation. https://github.com/gsi-upm/DrEWE, 2014. Accessed: 2015-03-31.

G. Cugola and A. Margara. Processing flows of information. *ACM Computing Surveys*, 44 (3):1–62, June 2012.

M. Czerwinski, E. Horvitz, and S. Wilhite. A diary study of task switching and interruptions. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, CHI '04, pages 175–182, New York, NY, USA, 2004. ACM.

G. De Francisci Morales, A. Gionis, and C. Lucchese. From chatter to headlines. In *Proceedings of the fifth ACM international conference on Web search and data mining*, page 153, New York, February 2012. ACM Press.

K. Dellschaft and S. Staab. On how to perform a gold standard based evaluation of ontology learningIn , *The Semantic Web - ISWC 2006*, volume 4273 of *Lecture Notes in Computer Science*, pages 228–241. Springer Berlin Heidelberg, 2006.

P. Dempsey. The teardown amazon echo digital personal assistant [teardown consumer electronics]. *Engineering Technology*, 10(2):88–89, March 2015.

L. Drumond and R. Girardi. A survey of ontology learning procedures. volume 427, 2008.

M. Eckert, F. Bry, S. Brodt, O. Poppe, and S. Hausmann. A CEP Babelfish: Languages for Complex Event Processing and Querying SurveyedIn , *Reasoning in Event-Based Distributed Systems*, volume 347 of *Studies in Computational Intelligence*, pages 47–70. Springer Berlin Heidelberg, 2011.

Elasticio. Elastic.io integrate once. connect many. `http://www.elastic.io/`, 2015. Accessed: 2015-03-31.

A. Erradi, P. Maheshwari, and V. Tosic. Policy-Driven Middleware for Self-adaptation of Web Services Compositions. In *Proceedings of the ACM/IFIP/USENIX 2006 International Conference on Middleware*, pages 62–80, 2006.

M. S. et al. Json-ld 1.0. `http://json-ld.org/spec/latest/json-ld/`, January 2014.

P. T. Eugster, P. A. Felber, R. Guerraoui, and A.-M. Kermarrec. The many faces of publish/subscribe. *ACM Comput. Surv.*, 35(2):114–131, June 2003.

J. I. Fernández-Villamor, C. A. Iglesias, and M. Garijo. First-Order Logic Rule Induction for Information Extraction in Web Resources. *International Journal of Artificial Intelligence Tools*, 21:1250032–1–,1250032–2, 2012.

C. Fürber and M. Hepp. Using SPARQL and SPIN for Data Quality Management on the Semantic Web. *Business Information Systems*, (1):35–46, 2010.

Y. Gil, P. Groth, and V. Ratnakar. Social task networks: Personal and collaborative task formulation and management in social networking sites. In *Proceedings of the AAAI Fall Symposium on Proactive Assistant Agents*, 2010.

D. Greenwood, M. Lyell, A. Mallya, and H. Suguri. The ieee fipa approach to integrating software agents and web services. In *Proceedings of the 6th international joint conference on Autonomous agents and multiagent systems*, AAMAS '07, pages 276:1–276:7, New York, NY, USA, 2007. ACM.

B. J. Grosz and S. Kraus. Collaborative plans for complex group action. *ARTIFICIAL INTELLIGENCE*, 86(2):269–357, 1996.

T. R. Gruber. A translation approach to portable ontology specifications. *Knowl. Acquis.*, 5(2):199–220, June 1993.

T. Gruber. Intelligence at the interface: Semantic technology and the consumer internet experience, 2009.

T. Gu, H. K. Pung, and D. Q. Zhang. A service-oriented middleware for building context-aware services. *Journal of Network and Computer Applications*, 28(1):1–18, January 2005.

M. Hazman, S. R. El-Beltagy, and A. Rafea. A survey of ontology learning approaches. *International Journal of Computer Applications*, 22:6, 2011.

M. Hepp. Goodrelations language reference. `http://www.heppnetz.de/ontologies/goodrelations/v1`, 2011. Version 1.0, , 01/10/2011.

M. N. Huhns and M. P. Singh. Personal assistants. *Internet Computing, IEEE*, 2(5):90–92, 1998.

Ifttt. IFTTT put the internet to work for you. `http://ifttt.com/`, 2015. Accessed: 2015-03-31.

M. Indiramma and K. Anandakumar. Collaborative decision making framework for multi-agent system. In *Computer and Communication Engineering, 2008. ICCCE 2008. International Conference on*, pages 1140 –1146, May 2008.

D. L. D. Ipiña. An ECA Rule-Matching Service for Simpler Development of Reactive Applications. *IEEE DSOnline*, 2, 2001.

itduzzit. Cloud integration - itduzzit. `http://cloud.itduzzit.com/`, 2015. Accessed: 2016-01-10.

P. Jaccard. *Etude comparative de la distribution florale dans une portion des Alpes et du Jura*. Impr. Corbaz, 1901.

C. Jacinto and C. Antunes. User-driven ontology learning from structured data. In *Computer and Information Science (ICIS), 2012 IEEE/ACIS 11th International Conference on*, pages 184–189. IEEE, 2012.

P. Jain, P. Z. Yeh, K. Verma, R. G. Vasquez, M. Damova, P. Hitzler, and A. P. Sheth. Contextual ontology alignment of lod with an upper ontology: A case study with proton. In *The Semantic Web: Research and Applications*, pages 80–92. Springer, 2011.

A. Jentzsch, R. Cyganiak, and C. Bizer. State of the lod cloud. `http://lod-cloud.net/state/`, 2011. Version 0.3, 09/19/2011.

J. J. Jung. Contextgrid: A contextual mashup-based collaborative browsing system. *Information Systems Frontiers*, 14(4):953–961, 2011.

D. Karger. The Semantic Web and End Users: What's Wrong and How to Fix It. *Internet Computing, IEEE*, 2014.

V. Kashyap. Design and creation of ontologies for environmental information retrieval. In *Proceedings of the 12th Workshop on Knowledge Acquisition, Modeling and Management*, pages 1–18. Citeseer, 1999.

J. H. Kietzmann, K. Hermkens, I. P. McCarthy, and B. S. Silvestre. Social media? Get serious! Understanding the functional building blocks of social media. *Business Horizons*, 54(3):241–251, May 2011.

M. Kifer. Rule interchange format: The frameworkIn , *Web Reasoning and Rule Systems*, volume 5341 of *Lecture Notes in Computer Science*, pages 1–11. Springer Berlin Heidelberg, 2008.

H. Labiod, A. Hossam, and C. D. Santis. *Wi-Fi, Bluetooth, Zigbee and WiMax.* Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2007.

M. S. LaTour and T. L. Henthorne. Informal interpersonal influence on purchasing agents' perceived risk. In *Proceedings of the 1987 Academy of Marketing Science (AMS) Annual Conference*, pages 371–375. Springer, 2015.

D. Lillis. *Internalising Interaction Protocols as First-Class Programming Elements in Multi Agent Systems.* PhD thesis, University College Dublin, 2012.

Z. Lin and K. Carley. Proactive or reactive: An analysis of the effect of agent style on organizational decision making performance. *Intelligent Systemt in Accounting, Finance and Management*, 1993.

J. N. Liu, Y.-L. He, E. H. Lim, and X.-Z. Wang. A new method for knowledge and information management domain ontology graph model. *Systems, Man, and Cybernetics: Systems, IEEE Transactions on*, 43(1):115–127, 2013.

C. M. MacKenzie, K. Laskey, F. McCabe, P. F. Brown, R. Metz, and B. A. Hamilton. Reference model for service oriented architecture 1.0. *OASIS Standard*, 12, 2006.

A. Martinez. *The Ultimate IFTTT Guide: Use The Web's Most Powerful Tool Like A Pro.* 2013.

B. D. Meester, D. Arndt, P. Bonte, J. Bhatti, W. Dereuddre, R. Verborgh, F. Ongenae, F. D. Turck, E. Mannens, and R. V. de Walle. Event-driven rule-based reasoning using eye. In *SSN-TC/OrdRing@ISWC*, volume 1488 of *CEUR Workshop Proceedings*, pages 75–86. CEUR-WS.org, 2015.

A. R. Meisel. *Optimze, Automate, and Outsource Everything In Your Life: How to Make Email, IFTTT, and Virtual Assistants Your Ultimate Productivity Weapons.* CreateSpace Independent Publishing Platform, 2014.

M. Montaner. A Taxonomy of Recommender Agents on the Internet. *Artificial Intelligence Review*, pages 285–330, 2003.

O. Moser, F. Rosenberg, and S. Dustdar. Non-intrusive monitoring and service adaptation for WS-BPEL. In *Proceeding of the 17th international conference on World Wide Web*, page 815, New York, 2008. ACM Press.

O. Moser, F. Rosenberg, and S. Dustdar. Event Driven Monitoring for Service Composition Infrastructures. In *The 11th International Conference on Web Information System Engineering 2010*, pages 38–51, Hong Kong, 2010.

J. P. Müller. Architectures and applications of intelligent agents: A survey. *The Knowledge Engineering Review*, 13(4):353–380, 1999.

M. A. Musen. Dimensions of knowledge sharing and reuse. *Comput. Biomed. Res.*, 25(5): 435–467, October 1992.

K. Myers, P. Berry, J. Blythe, K. Conley, M. Gervasio, D. L. McGuinness, D. Morley, A. Pfeffer, M. Pollack, and M. Tambe. An intelligent personal assistant for task and time management. *AI Magazine*, 28(2):47, 2007.

A. Nikolov and E. Motta. Capturing emerging relations between schema ontologies on the web of data. 2010.

D. A. Norman. How might people interact with agents. *Commun. ACM*, 37:68–71, July 1994.

M. O'Connor, H. Knublauch, S. Tu, B. Grosof, M. Dean, W. Grosso, and M. Musen. Supporting Rule System Interoperability on the Semantic Web with SWRLIn , *The Semantic Web – ISWC 2005*, volume 3729 of *Lecture Notes in Computer Science*, pages 974–986. Springer Berlin Heidelberg, 2005.

M. OĆonnor, M. Dean, H. Boley, and A. Das. SWRL2 - Ruleml. `http://wiki.ruleml.org/index.php/SWRL2`, 2012. Accessed: 2015-01-10.

I. Ohmukai, H. Takeda, and M. Miki. A proposal of the person-centered approach for personal task management. *2003 Symposium on Applications and the Internet, 2003. Proceedings.*, pages 234–240, 2003.

K. A. Olsen and A. Malizia. Automated personal assistants. *Computer*, 44(11):112, 110–111, 2011.

OMG. Production Rule Representation. Technical report, Object Managment Group, 2009. Accessed: 2016-01-10.

Onx. On{x} automate your life. `https://www.onx.ms/`, 2015. Accessed: 2015-03-31.

K. Opasjumruskit, J. Expósito, and B. König-Ries. MERCURY: User Centric Device and Service Processing–Demo paper. *ABIS 2012*, pages 2–5, 2012.

L. Ouyang, B. Zou, M. Qu, and C. Zhang. A method of ontology evaluation based on coverage, cohesion and coupling. In *2011 Eighth International Conference on Fuzzy Systems and Knowledge Discovery*, pages 2451–2455. IEEE, July 2011.

K. Parks and D. Watkins. *How to Automate Your Way to Freedom.* 2012. ebook.

A. Paschke and A. Kozlenkov. Rule-Based Event Processing and Reaction Rules. In *Proceedings of the 2009 International Symposium on Rule Interchange and Applications*, pages 53–66, 2009.

A. Paschke and P. Vincent. A reference architecture for Event Processing. In *Proceedings of the Third ACM International Conference on Distributed Event-Based Systems*, page 1, New York, 2009. ACM Press.

A. Paschke, P. Vincent, and F. Springer. Standards for Complex Event Processing and Reaction RulesIn , *Rule-Based Modeling and Computing on the Semantic Web*, pages 128–139. Springer-Verlag, 2011.

E. Pignotti and P. Edwards. Using Web Services and Policies within a Social Platform to Support Collaborative Research. In *Working Notes of AAAI 2012 Stanford Spring Symposium on Intelligent Web Services Meet Social Computing (March 2012)*, pages 64–69, 2012.

A. Pintus, D. Carboni, and A. Piras. Paraimpu: a platform for a social web of things. In *Proceedings of the 21st international conference companion on World Wide Web*, pages 401–404, 2012.

A. Pokahr and L. Braubach. From a research to an industry-strength agent platform: Jadex v2. *Business Services: Konzepte, Technologien, Anwendungen. 9. Internationale Tagung Wirtschaftsinformatik*, pages 769–780, 2009.

J. F. S. Rada, C. A. I. Fernandez, and M. C. Barrios. Maia: an event-based modular architecture for intelligent agents. In *IEEE/WIC/ACM International Conference on Intelligent Agent Technology (WIC 2014)*, pages 87–94, Agosto 2014.

Y. Raimond and S. Abdallah. The Event Ontology. Technical report, Technical report, 2007. http://motools. sourceforge., 2012. Accessed: 2015-03-31.

Y. Raimond, S. A. Abdallah, M. B. Sandler, and F. Giasson. The music ontology. In *ISMIR*, pages 417–422. Citeseer, 2007.

Y. Raimond, T. Gangler, F. Giasson, K. Jacobson, G. Fazekas, S. Reinhardt, and A. Passant. The Music Ontology - Specification. `http://purl.org/ontology/mo/`, 2013. Version 2.1.5, , 22/07/2012.

A. S. Rao, M. P. Georgeff, et al. Bdi agents: From theory to practice. In *Proceedings of the first international conference on multi-agent systems (ICMAS-95)*, pages 312–319. San Francisco, 1995.

V. Ricquebourg, D. Durand, D. Menga, B. Marhic, L. Delahoche, and C. Logé. Context inferring in the Smart Home: An SWRL approach. In *21st International Conference on Advanced Information Networking and Applications Workshops, 2007*, number iv, pages 290–295, Niagara Falls, 2007.

J. Ruan and Y. Yang. Assess Content Comprehensiveness of Ontologies. *2010 Second International Conference on Computer Modeling and Simulation*, pages 536–539, January 2010.

A. Ruttenberg, J. Carroll, and M. Krotzsch. Punning - OWL. `http://www.w3.org/2007/OWL/wiki/Punning`, 2008. Version 10/07/2008.

M. Sabou, C. Wroe, C. Goble, and H. Stuckenschmidt. Learning domain ontologies for semantic web service descriptions. *Web Semantics: Science, Services and Agents on the World Wide Web*, 3(4):340 – 365, 2005. World Wide Web Conference 2005——Semantic Web TrackWorld Wide Web Conference 2005——Semantic Web Track.

F. Sadri. Ambient intelligence: A survey. *ACM Computing Surveys*, 43(4):1–66, October 2011.

D. Sánchez. *Domain Ontology Learning from the Web: An Unsupervised, Automatic and Domain Independent Approach.* AV Akademikerverlag, 2012.

D. Sánchez and A. Moreno. Creating ontologies from web documents. *Recent Advances in Artificial Intelligence Research and Development. IOS Press*, 113:11–18, 2004.

A. Seaborne. Sparql results in json. `http://www.w3.org/TR/sparql11-results-json/`, January 2011.

M. Shamsfard and A. A. Barforoush. The state of the art in ontology learning: a framework for comparison. *Knowledge Engineering Review*, 18:293–316, 2003.

S. Shapiro, S. Sardina, J. Thangarajah, L. Cavedon, and L. Padgham. Revising conflicting intention sets in bdi agents. In *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems - Volume 2*, AAMAS '12, pages 1081–1088, Richland, SC, 2012. International Foundation for Autonomous Agents and Multiagent Systems.

R. Shaw, R. Troncy, and L. Hardman. LODE: Linking Open Descriptions of Events. *School of Information*, 2009.

V. K. Singh and R. Jain. Structural analysis of the emerging event-web. In *Proceedings of the 19th International Conference on World Wide Web*, pages 11–83, New York, April 2010. ACM Press.

D. Spohr, P. Cimiano, and J. M. Crae. Using SPIN to Formalise Accounting Regulations on the Semantic Web. In *International Workshop on Finance and Economics on the Semantic Web (FEOSW 2012)*, pages 1–15, 2012.

D. Steiner. Fipa: Foundation for intelligent physical agents - das aktuelle schlagwort. *KI*, 12(3):38, 1998.

SWRL. SWRL: A Semantic Web Rule Language Combining OWL and RuleML. `http://www.w3.org/Submission/SWRL/`, 2005. Accessed: 2015-06-10.

Tasker. Tasker total automation for android. `http://tasker.dinglisch.net/`, 2015. Accessed: 2015-03-31.

I. Tiddi, N. Mustapha, Y. Vanrompay, and M.-A. Aufaure. Ontology learning from open linked data and web snippetsIn , *On the Move to Meaningful Internet Systems: OTM 2012 Workshops*, volume 7567 of *Lecture Notes in Computer Science*, pages 434–443. Springer Berlin Heidelberg, 2012.

Topbraid. TopBraid Suite 3.5.0 changelog.

M. Van Kleek, B. Moore, D. R. Karger, P. André, and M. Schraefel. Atomate it! end-user context-sensitive automation using heterogeneous information sources on the web. In *Proceedings of the 19th international conference on World wide web - WWW '10*, page 951, New York, New York, USA, 2010. ACM Press.

J. Vassileva. A review of organizational structures of personal information management. *Management*, pages 1–19, 2008.

R. Verborgh and J. De Roo. Drawing conclusions from linked data on the web: The eye reasoner. *Software, IEEE*, 32(3):23–27, May 2015.

K. Vladimir, I. Budiselić, and S. Srbljić. Consumerized and peer-tutored service composition. *Expert Systems with Applications*, 42(3):1028 – 1038, 2015.

P. Wallis, R. Ronnquist, D. Jarvis, and A. Lucas. The automated wingman - using jack intelligent agents for unmanned autonomous vehicles. In *Aerospace Conference Proceedings*, volume 5, pages 5–2615–5–2622. IEEE, 2002.

Wappwolf. Automate your dropbox. `http://wappwolf.com/`, 2015. Accessed: 2016-01-10.

Webee. Webee experience, connected. `http://www.webeeuniverse.com/`, 2015. Accessed: 2015-03-31.

U. Westermann and R. Jain. Toward a Common Event Model for Multimedia Applications. *IEEE Multimedia*, 14(1):19–29, 2007.

WigWag. WigWag smart starts with a brain. `http://www.wigwag.com/`, 2015. Accessed: 2015-03-31.

P. J. Windley. *The Live Web: Building Event-Based Connections in the Cloud.* Course Technology, 2011a.

P. Windley. *The Live Web: Building Event-Based Connections in the Cloud.* Course Technology, 2011b.

W. Wong, W. Liu, and M. Bennamoun. Ontology learning from text: A look back and into the future. *ACM Comput. Surv.*, 44(4):20:1–20:36, sep 2012.

M. J. Wooldridge and N. R. Jennings. *Intelligent agents.* Springer-Verlag, 1995.

S.-Y. Yang. Developing an energy-saving and case-based reasoning information agent with web service andontology techniques. *Expert Systems with Applications*, 40(9):3351 – 3369, 2013.

N. Yorke-Smith, S. Saadati, K. L. Myers, and D. N. Morley. The design of a proactive personal agent for task management. *International Journal on Artificial Intelligence Tools*, 21(01):1250004, 2012.

J. Yu, B. Benatallah, F. Casati, and F. Daniel. Understanding Mashup Development. *IEEE Internet Computing*, 12(5):44–52, September 2008.

Y. Yueh, D. Chiu, H. fung Leung, and P. Hung. A virtual travel agent system for m-tourism with semantic web service based design and implementation. In *Advanced Information Networking and Applications, 2007. AINA '07. 21st International Conference on*, pages 142–149, May 2007.

Zapier. Zapier the best apps. better together. `http://zapier.com/`, 2015. Accessed: 2015-03-31.

List of Figures

List of Tables

APPENDIX A

Publications

The results of this thesis have produced a number scientific publications in journals and in conference proceedings. The list of those publications is shown below:

A.1 Journal Articles

- Miguel Coronado, and Carlos A. Iglesias. Task Automation Services: Automation for the masses. IEEE Internet Computing, 99:52-58, 2015. ISSN 1089-7801. doi: 10.1109/MIC.2015.73. **Impact Factor** (2014): 1.71 **Q1**.

- Miguel Coronado, Carlos A. Iglesias, and Emilio Serrano. Modelling Rules for Automating the Evented Web by Semantic Technologies. Expert Systems With Applications , 37:7979-7990, 2015. ISSN 0957-4174. doi: 10.1016/j.eswa.2015.06.031. **Impact Factor** (2014): 2.240 **Q1**.

- Juan F. Sanchez-Rada, Carlos A. Iglesias, and Miguel Coronado. MAIA: An Event-based Modular Architecture for Intelligent Agents, VV(1):pp-pp, 2016. ISSN 1570-1263. SJC Factor (2014): 0.45 **Q2** (in press).

- Miguel Coronado, Carlos A. Iglesias, Alejandro López, and Mercedes Garigo. TutorGSI: aplicación de tecnologías de bots a entorno LMS. Revista de Educacion a distancia (RED), 28:1-12, 2011. ISSN 1578-7680.

A.2 Conference Proceedings

- Miguel Coronado, Alberto Mardomingo, and Carlos A. Iglesias. A Personal Agents Hybrid Architecture for Question Answering featuring Social Dialog. In International Symposium on INnovations in Intelligent SysTems and Applications (INISTA), 2015 International Conference on, pages 1-8, September 2015. doi: 10.1109/INISTA.2015.7276780.

- Miguel Coronado, Ralf Bruns, Jurgen Dunkel, and Sebastian Stipkovic. Context-awareness in Task Automation Services by Distributed Event Processing. In Proceedings of the 15th Internacional Conference of Web Information System Engineering (WISE 2014), pages 190-203, September 2014. ISBN 0302-9743. doi: 10.1007/978-3-319-20370-6_15.

- Juan F. Sanchez-Rada, Carlos A. Iglesias, and Miguel Coronado. MAIA: An Event-based Modular Architecture for Intelligent Agents. In Web Intelligence (WI) and Intelligent Agent Technologies (IAT), 2014 IEEE/WIC/ACM International Joint Conferences on (Volume:3), pages 87-94. IEEE, 2012. INSPEC: 14686910. doi: 10.1109/WI-IAT.2014.154.

- Miguel Coronado, Felipe Echanique, and Carlos A Iglesias. Personal Agents for managing social notifications in an e-learning environment. In Proceedings of 5th International Work Conference of the interplay between natural and artificial computation (IWINAC 2013), Palmanova 2013.

APPENDIX B

The Evented WEb Ontology Specification

One of the contributions of the thesis is the EWE ontology - a formal specification of the reference model for describing Task Automation Services. The following appendix contains a summarized version of the specification. In comparison, the full specification available on-line is a more structured document with a greater number links, and back-references that facilitate improved specification browsing in the web environment. Moreover, the on-line version is a live specification that will be updated any time changes are approved into the EWE specification. The summarized version presented in the appendix contains all the information required for the proposed modelling process as does its web counterpart. The following appendix is primary a supplement for Chapter 3.

EWE Ontology Specification

V1.1 - 10 Jan 2016

This version: http://gsi.dit.upm.es/ontologies/ewe/1.0 (RDF/XML, HTML)
Latest version: http://gsi.dit.upm.es/ontologies/ewe
Editors: Miguel Coronado, Carlos A. Iglesias
Authors: Miguel Coronado
Contributors: See acknowledgements

Abstract

Evented **WE**b Ontology (**EWE**) is a standardized data schema (also referred as "ontology" or "vocabulary") designed to describe elements within Task Automation Services enabling rule interoperability. The following document contains the description of ontology and instructions how to connect it with descriptions of other resources.

Table of Contents

1 Introduction

The following specification is a formal description of metadata schema proposal that can be applied to data gathered in the so-called Task Automation Services. The goal of the following section is to introduce both Semantic Web and Task Automation Services to the topic and goals of the ontology and provide the basic knowledge to comprehend the technical part of the specification.

1.1 Task Automation Services

A number of now prominent web sites, mobile and desktop applications feature rule-based task automation. Typically, these services provide users the ability to define which action should be executed when some event is triggered. Some examples of this simple task automation could be *“When I am mentioned in Twitter, send me an email”*, *“When I reach 500 meters of this place, check in in Foursquare”*, or *“Turn Bluetooth on when I leave work”*. We call them Task Automation Service (TAS). Some TASs allow users to share the rules they have developed, so that other users can reuse these tools and adapt them to their own preferences.

Task Automation is a rising area, recently lots different web services and mobile-apps focus their business on this topic. Although the concept is not new, several changes on the state of technology supports the success of these services and applications. Among them the massive publishing of third-party APIs on the Cloud, providing access to their services is a key factor that unchained this mushrooming.

We could find several details that difference these services to each other. Some are web services and some are mobile-apps, so they distinguish their scope. Those smartphone based focus on those capabilities that are provided by the device that cannot be addressed in a web environment: switching on/of BT under certain conditions, automatically connect the wireless receiver when we reach some place, automation based on positioning, based on the level of the battery etc. At the time of writing these lines, there is no Task Automation Service that cover both a web service and a smartphone application, i.e. that let you define rules that mix both worlds, triggered by events from the web that execute tasks on the mobile phone, or vice versa.

We can also classify them according to the target audience. Some of them focus on the user, on automation of their repetitive task, e.g. changing Twitter's profile picture when we change Facebook's, or texting my mother that I'm running out of battery when its level is below 10%. For of the mobile based TAS are of this type. Some other, are more specific and focus on business tasks. We can perceive this by examining the category of the services (so-called channels) they integrate. Also, depending on the process used to create the rule, we can infer the target, this is, those that provide a rule language are focused on programmers.

Some of the most popular Task Automation Services are:

- ifttt
- Zapier
- CouldWork
- Onx
- Atooma
- AutomateIt

1.2 The Semantic Web

The Semantic Web is a W3C initiative that aims to introduce rich metadata to the current Web and provide machine readable and processable data as a supplement to human-readable Web.

Semantic Web is a mature domain that has been in research phase for many years and with the increasing amount of commercial interest and emerging products is starting to gain appreciation and popularity as one of the rising trends for the future Internet.

One of the corner stores of the Semantic Web is research on inerlinkable and interoperable data schemas for information published online. Those schemas are often referred to as ontologies or vocabularies. In order to facilitate the concept of ontologies that lead to a truly interoperable Web of Data, W3C has proposed a series of technologies such as RDF and OWL. **EWE** uses those technologies and the research that comes within to propose an ontology set in the domain of Task Automation Services.

1.3 What is EWE for?

The goals of the EWE ontology to achieve are:

- Enable to publish raw data from Task Automation Services (Rules and Channels) online and in compliance with current and future Internet trends
- Enable Rule interoperability
- Provide a base vocabulary for building domain specific vocabularies e.g. Twitter Task Ontology or Evernote Task Ontology

2. EWE ontology at a glance

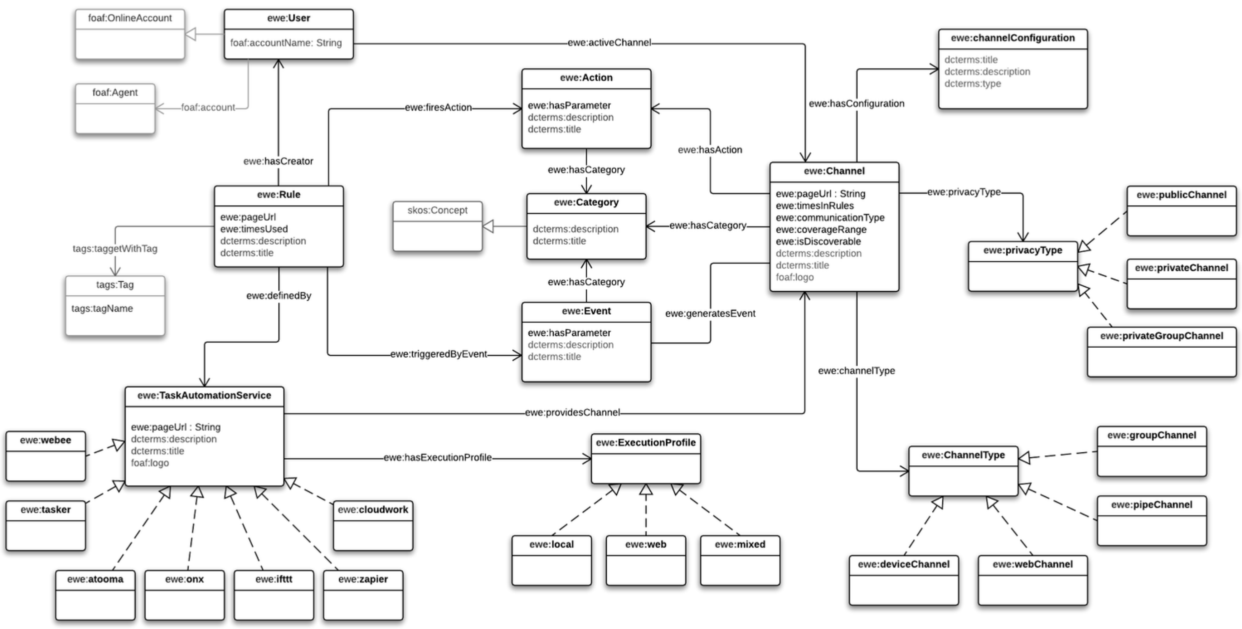
An alphabetical index of EWE terms, by class (concepts) and by property (relationships, attributes), are given below. All the terms are hyperlinked to their detailed description for quick reference.

Classes: | Action | Agent | Channel | Event | InputParameter | OnlineAccount | OutputParameter | Parameter | Rule | Tag | Tagging | TaskAutomationService | User |

Properties: | created | generatesEvent | hasActiveChannel | hasCreator | hasInputParameter | hasOutputParameter | hasParameter | isGeneratedBy | isProvidedBy | isUsedIn | logo | page | providesAction | supportedBy | supportsChannel | tag | taggedWithTag | timesUsed | triggeredBy | uses |

3. EWE ontology overview

The EWE UML diagram presented below shows connections between classes that implement the data model of Task Automation Services.



UML Class Diagram for the EWE Ontology (high resolution version: PNG)

3.1. Referenced Namespaces

A common practice in Semantic Web domain is to take advantage of the work previously done and model new domains with the use of widely known and well established vocabularies. This practice simplifies the vocabulary pool, allows to establish naming standards and implement better interoperability and data portability across potentially very different systems.

We attempted to follow those rules, thus some of the classes and properties defined in this specification have references to properties from other ontologies to indicate point of connection. Those referenced ontologies are described with the following namespaces:

Prefix	Namespace	Specification/ Usage Description
dcterms	http://purl.org/dc/terms/	Dublin Core Terms ontology is a core ontology that defines a number of very generic properties such as title, description, value, etc. Those are used across many classes in the EWE ontology.
foaf	http://xmlns.com/foaf/0.1/	Friend Of A Friend ontology. Within EWE it is used to model concepts related to user account and the digital personification of a human being.
skos	http://www.w3.org/2004/02/skos/core#	Simple Knowledge Organization System is a common data model for knowledge organization systems such as thesauri, classification schemes, subject heading systems and taxonomies. Within EWE, it is used to describe different categories of Channels and Events.
tags	http://www.holygoat.co.uk/owl/redwood/0.1/tags/	Newman's Tags ontology. The concepts of this ontology are used as a base model tags.
spin	http://spinrdf.org/spin#	SPARQL Inferencing Notation. EWE's rules are described using SPIN SPARQL Syntax, an RDF representation of the semantic web query language SPARQL.

The next diagram summarizes relation between external vocabularies and their relation to EWE.

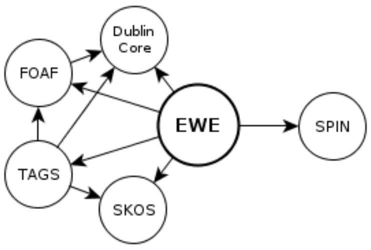


Diagram that shows relation among external vocabularies and EWE (PNG).

3.2. EWE Channel example

A very basic example below shows a single Channel described using EWE vocabulary. It defines a new subclass of Channel that outlines how GoogleTalk Service works. As defined below, Gmail Channel generates two events and provide one single action. These are *"Any* *new email"* and *"New email from"* events and *"Send an email"* action (their RDF description is not shown in the example below for sake of simplicity).

This is a real example of class definition scrapped from ifttt.com (channel definition provided here). Note that the current version of channel description at ifttt.com may differ from the description shown here, due to ifttt is in continuous expansion, remodeling their channels, so new events or actions may have been added since this example was written down .

```
<owl:Class rdf:about="https://ifttt.com/gmail">
  <rdfs:subClassOf rdf:resource="http://gsi.dit.upm.es/ontologies/ewe/ns#Channel"/>

  <!-- Administrative properties -->
  <dcterms:title>Gmail</dcterms:title>
  <dcterms:description>
    Gmail is a free, advertising-supported webmail, POP3, and IMAP service provided by Google.
  </dcterms:description>
  <foaf:logo>https://ifttt.com/images/channels/gmail_lrg.png</foaf:logo>

  <!-- Categorization -->
  <ewe:hasCategory rdfs:resource="http://gsi.dit.upm.es/ontologies/ewe/ns#email">

  <!-- Functionalities -->
  <ewe:generatesEvent rdf:resource="https://ifttt.com/channels/gmail/triggers/85"/>
  <ewe:generatesEvent rdf:resource="https://ifttt.com/channels/gmail/triggers/86"/>
  <ewe:hasAction rdf:resource="https://ifttt.com/channels/gmail/actions/34"/>
</owl:Class>
```

In the former example events and actions are included as external references. This is the preferred way for describing channels, since it is easier to read, and offers a more modular view of the model. However, as in any other RDF graph, RDF entities can be nested, thus we can include the Event or Action definition nested within the Channel definition. We could even add them as *black entities or nodes* if there is no need to reference them from the outside (this is without relating them to the Channel that defines them), although not common and it is discouraged.

The example below provides the description of the *"New Email from"* Event referenced at the Gmail Channel definition from the example above. The event shown presents one input parameter -the email address of the sender- and three output parameters -the email address of the sender, the subject of the email, and the body of the message in plain text.

In this case, parameters are included as nested elements instead of being referenced as external resources. Although parameters have not been given an ID (for sake of simplicity on the example) we encourage you to do so. Moreover, the reference-to-external-resources is also an acceptable approach (once more, we did not use it here to save the reader from following multiple link from the referral to the definition).

```
<owl:Class rdf:about="https://ifttt.com/channels/gmail/triggers/86">
  <rdf:type rdf:resource="http://www.semanticweb.org/ontologies/2012/9/ewe.owl#Event"/>
  <dcterms:title>New email from</dcterms:title>
  <dcterms:description>
    This Trigger fires every time a new email arrives in your inbox from the address you specify.
  </dcterms:description>

  <!-- Input Parameters -->
  <ewe:hasInputParameter>
    <ewe:InputParameter>
      <dcterms:title>EmailAddress</dcterms:title>
    </ewe:InputParameter>
  </ewe:hasInputParameter>

  <!-- Output Parameters -->
  <ewe:hasOutputParameter>
    <ewe:OutputParameter>
      <dcterms:title>FromAddress</dcterms:title>
      <dcterms:description>Email address of sender.</dcterms:description>
    </ewe:OutputParameter>
  </ewe:hasOutputParameter>
  <ewe:hasOutputParameter>
    <ewe:OutputParameter>
      <dcterms:title>Subject</dcterms:title>
      <dcterms:description>Email subject line.</dcterms:description>
    </ewe:OutputParameter>
  </ewe:hasOutputParameter>
  <ewe:hasOutputParameter>
    <ewe:OutputParameter>
      <dcterms:title>BodyPlain</dcterms:title>
      <dcterms:description>Plain text email body.</dcterms:description>
    </ewe:OutputParameter>
  </ewe:hasOutputParameter>

</owl:Class>
```

Just one more detail, the former example uses the properties `ewe:hasInputParameter` and `ewe:hasOutputParameter` to reference the Input and Output Parameters. However, the parent `hasParameter` could have also been used with `OutputParameter` or `InputParameter`. When the inference engine is available, both approaches are equivalent and any query that matches one of them will also match the other.

Nevertheless, when it is not, the procedure shown is more explicit, thus preferred.

3.3. SPARQL Construct EWE Rule example

The example below shows how is a EWE rule defined using a `CONSTRUCT` query.

The rule in the example sends the user a new chat message thought the GoogleTalk channel every time a new email is received. The content of the chat message specify the senders email address, so the user can decide if it is important or not.

In detail, the `WHERE` graph selects all events whose title is *"Any new mail"*, that also have an output

```
CONSTRUCT {
    ?action a ewe:Action .
    ?action dcterms:title "New chat message" .
    ?action ewe:hasParameter ?iParam .
    ?iParam dcterms:title "message" .
    ?iParam dcterms:value ?message .
    <https://ifttt.com#GoogleTalk;> ewe:providesAction ?action .
}
WHERE {
    ?event a ewe:Event .
    ?event dcterms:title "Any new email" .
    ?event ewe:hasOutputParameter ?oParam .
    <https://ifttt.com/Gmail> ewe:generatesEvent ?event .
    ?oParam dcterms:title "FromAddress" .
    ?oParam dcterms:value ?oParamFrom .
    BIND (fn:concat("You have received a new message from ", ?oParamFrom) AS ?message) .
    BIND (URI("http://example.org/action1") AS ?action) .
    BIND (URI("http://example.org/param1") AS ?iParam) .
}
```

The former rule is written using a SPARQL `CONSTRUCT` query. It can be transformed into a RDF format using the SPIN language. This provide some advantages such as the capability of store the rule in the RDF endpoint with the Channels, Events and Actions. It is executable code, since SPIN libraries provide functions to transform it back to the defining SPARQL query and also to execute it directly. In addition, since the rule is loaded in the endpoint, it can be queried to get rules that meet a certain criteria.

The example below shows an SPARQL query to extract all the EWE rules stored that connects Gmail's events to Google talk's actions.

```
SELECT * WHERE {
    _:srule rdf:type ewe:SpinRule ;
            dcterms:title ?ruletitle ;
            spin:rule _:aux1 .
    _:aux1 sp:where _:aux2 .
    _:aux2 rdf:rest*/rdf:first ?triplet_w .
    ?triplet_w sp:predicate ewe:generatesEvent ;
               <sp:subject https://ifttt.com/gmail> .
    _:aux1 sp:templates _:aux3 .
    _:aux3 rdf:rest*/rdf:first ?triplet_t .
    ?triplet_t sp:predicate ewe:providesAction ;
               sp:subject <https://ifttt.com/gtalk> .

}
```

3.4. Enhance channel search example

Modeling Task Automation Service assets using EWE Ontology provides a great deal of expressiveness, enhancing search queries of Channels and Rules. The example below shows several meaningful queries that can be done:

Query	SPARQL Query	Results

How many channels are defined by each Task Automation Service?	`SELECT ?tas (COUNT(?channel) AS ?channels)`
	`WHERE {`
	`?channel rdfs:subClassOf ewe:Channel .`
	`?channel ewe:supportedBy ?tas`
	`}`
	`GROUP BY ?tas`
		Zapier (141), Tasker (22), Ifttt (59)

Which channel categories are defined?	`SELECT DISTINCT ?category`
	`WHERE {`
	`  ?channel rdfs:subClassOf ewe:Channel .`
	`  ?channel ewe:hasCategory ?category`
	`}`
		Event Management, CRM, Phone, Developer Tools, Email Marketing, Social and 9 more.

Which channels are categorized as social?	`SELECT ?channelName WHERE { ?channel rdfs:subClassOf ewe:Channel . ?channel ewe:hasCategory ewe:Social . ?channel dcterms:title ?channelName }`	Wordpress, chatter, buffer, facebook, twitter, and 7 more
Which channels are categorized as ’developer tools’ and ’Phone’?	`SELECT ?channelName WHERE { ?channel rdfs:subClassOf ewe:Channel . ?channel ewe:hasCategory ewe:Phone . ?channel ewe:hasCategory ewe:Developer_Tools . ?channel dcterms:title ?channelName }`	Twillio and Opsgenie
Which channels’ events can be used without configuration?	`SELECT DISTINCT ?channelName ?eventName WHERE { ?channel rdfs:subClassOf ewe:Channel . ?channel dcterms:title ?channelName . ?channel ewe:generatesEvent ?event . ?event dcterms:title ?eventName . FILTER NOT EXISTS { ?event ewe:hasInputParameter ?inp } } GROUP BY ?channelName ?eventName`	Paypal’s Succecc Sale, Olark’s New Message, Google-doc’s New Document and 324 more.
How many actions can be executed by means of the Tasker TAS?	`SELECT (COUNT(?action) AS ?actsProvided) WHERE { ?channel ewe:hasAction ?action . ?channel ewe:supportedBy ewe:Tasker } GROUP BY ?channel`	204 actions

4. Cross-reference for EWE classes and properties

Below see a comprehensive list of all EWE classes, properties and their descriptions.

Classes and Properties (full detail)

Classes

Class: ewe:Action

Action - Action class defines an operation or proccess provided by a Service. Actions produce effects whose nature depend on the action's nature (e.g. producing a log message, modifying a public or private state on a server, switching on a ligth, etc.). Actions are on the right part of the Rule. Web services, actuators, and smartphone apps can provede actions when modeled as Services

Used with:	providesAction
Sub class of	owl:Thing

[#] [back to top]

Class: foaf:Agent

Agent - It describes an Agent using the FOAF ontology

Sub class of	owl:Thing

[#] [back to top]

Class: ewe:Channel

Channel - It defines individuals that either generate Events, provide Actions or both. In the context we refer, Channel mostly defines Web Services. However, sensors and actuators are also described as channels, thus they produce events or provide actions (e.g. a GPS device programmed to generate alerts when it is near certain locations).

Properties include:	providesAction generatesEvent

Used with:	hasActiveChannel uses supportsChannel
Sub class of	owl:Thing

[#] [back to top]

Class: ewe:Event

Event - Event class defines a particular ocurrence of a process, they are generated by a particular Service (e.g. new_chat_message events are generated by GoogleTalk service). Events have no duration. Changes on the state of a system or a sensor, can be modeled as events (the change in the state triggers the Event generation).

Used with:	generatesEvent triggeredBy
Sub class of	owl:Thing

[#] [back to top]

Class: ewe:InputParameter

InputParameter - InputParameters are configuration parameters that can be used to define the specific behavior of Events or Actions. Event's triggering conditions are set using InputParameters. Action's execution results is set using InputParameters.

Used with:	hasInputParameter
Sub class of	Parameter

[#] [back to top]

Class: foaf:OnlineAccount

OnlineAccount - It describes an OnlineAccount using FOAF ontology. The OnlineAccount class represents the provision of some form of online service, by some party (indicated indirectly via a accountServiceHomepage) to some Agent. The account property of the agent is used to indicate accounts that are associated with the agent.

Sub class of	owl:Thing
Has sub class	User

[#] [back to top]

Class: ewe:OutputParameter

OutputParameter - OutputParameters are more common in Events than in Actions. They specify the output arguments from an Event ocurrences. For instance, the timestamp of a chatmessage, the author of a tweet, the subject of an email, etc. The value of an output parameter is often bound to the input parameter while connected in a Rule.

Used with:	hasOutputParameter
Sub class of	Parameter

[#] [back to top]

Class: ewe:Parameter

Parameter - Parameters provide additional data that is either consumed or generated by Actions or Services. Depending on the nature of the Paremeter they can be either InputParameter or OutputParameter. The value of a Parameter is given by the property dcterms:value, and its name by the property dcterms:title. The property dcterms:description can also provide an overview of its meaning and usage.

Properties include:	value
Used with:	hasParameter
Sub class of	owl:Thing
Has sub class	InputParameter OutputParameter

[#] [back to top]

Class: ewe:Rule

Rule - Rule defines an ECA rule, triggered by an Event that means the execution of an Action. It defines particular interconections among instances of event and action, that includes the configuration parameters set for both of them. Rules are defined as a SPARQL construct query by means of the property spin:rule, using the SPIN language.

Properties include:	hasCreator timesUsed triggeredBy uses
Sub class of	owl:Thing

[#] [back to top]

Class: tags:Tag

Tag - It describes a certain tag annotation using the tags ontology

Used with:	taggedWithTag
Sub class of	owl:Thing

[#] [back to top]

Class: tags:Tagging

Tagging - It describes a set of tags applied to a certain resource by a certaing agent. Class from Tagging ontology.

Used with:	tag

[#] [back to top]

Class: ewe:TaskAutomationService

TaskAutomationService - This describes a Task Automation Service for the Evented Rules defined in EWE.

Properties include:	supportsChannel

[#] [back to top]

Class: ewe:User

User - It defines a User who define some AppRules among the services he has account in.

Properties include:	hasActiveChannel
Used with:	hasCreator
Sub class of	OnlineAccount foaf:OnlineAccount

[#] [back to top]

Properties

Property: dcterms:created

created - It indicates the data at wich a Rule was created.

Domain:	owl:Thing or ewe:Rule

[#] [back to top]

Property: ewe:generatesEvent

generatesEvent - Describes the event that can be generated by the Service (e.g. Weather forecast service may generate rain alert events).

Domain:	Channel
Range:	Event
Has inverse property	isGeneratedBy

[#] [back to top]

Property: ewe:hasActiveChannel

hasActiveService - It states that the User has the target service activated. This means, it has an acoount on that service and so he can use it.

Domain:	User
Range:	Channel

[#] [back to top]

Property: ewe:hasCreator

hasCreator - The creator of the Rule. It enables user profiling and authoring.

Domain:	Rule
Range:	User

[#] [back to top]

Property: ewe:hasInputParameter

hasInputParameter - Links Actions or Events to an Input-Parameter.

Domain:	ewe:Action or ewe:Event
Range:	InputParameter
Sub property of	hasParameter

[#] [back to top]

Property: ewe:hasOutputParameter

hasOutputParameter - Links Actions or Events to an Output-Parameter.

Domain:	ewe:Action or ewe:Event
Range:	OutputParameter
Sub property of	hasParameter

[#] [back to top]

Property: ewe:hasParameter

hasParameter - Links Actions or Events to a Parameter.

Domain:	owl:Thing or ewe:Action or ewe:Event
Range:	Parameter
Has sub property	hasOutputParameter hasInputParameter

[#] [back to top]

Property: ewe:isGeneratedBy

isGeneratedBy - It points out the Service that may generate the Event (e.g new chat message may be generated by Google Talk Service)

Inverse property of	the anonymous defined property with the label '*generatesEvent*' (**Object Property**)

[#] [back to top]

Property: ewe:isProvidedBy

isProvidedBy - It points out a Service that provides the Action (e.g. send new blog entry is provided by a RSS channel)

Inverse property of	the anonymous defined property with the label '*provicesAction*' (**Object Property**)

[#] [back to top]

Property: ewe:isUsedIn

isUsedIn - Describes an AppRule that is using any of the Events that generates or Actions that provides the Service

Inverse property of	the anonymous defined property with the label '*uses*' (**Object Property**)

[#] [back to top]

Property: foaf:logo

logo - it describes the logo of a given Service

Domain:	owl:Thing or ewe:TaskAutomationService or ewe:Channel

[#] [back to top]

Property: foaf:page

page - Describes the original web resource that the class models or referes to. It is described using the FOAF ontology, accordint to which, this points out to a document it is about.

Domain:	owl:Thing or ewe:Rule or ewe:TaskAutomationService or ewe:Channel

[#] [back to top]

Property: ewe:providesAction

provicesAction - Describe an action provided by the service (e.g. Email services provides send new email action)

Domain:	Channel
Range:	Action
Has inverse property	isProvidedBy

[#] [back to top]

Property: ewe:supportedBy

supportedBy - It indicates that a channel is supported by the TaskAutomationService given. It is said that a Channel is supported by a certain TaskAutomationService when rules that involve use of Events or Actions generated or provided by that Channel can be executed on the TaskAutomationService. It is used to build the list of TaskAutomationServices that support a certain channel.

Inverse property of	the anonymous defined property with the label '*supportsChannel*' (**Object Property**)

[#] [back to top]

Property: ewe:supportsChannel

supportsChannel - It indicates that a TaskAutomationService supports the Channel given. It is said that a TaskAutomationService supports a certain Channel when rules that involve use of Events or Actions generated or provided by that Channel can be executed on the TaskAutomationService. It is used to define the list of Channels that a instance of TaskAutomationService supports.

Domain:	TaskAutomationService
Range:	Channel
Has inverse property	supportedBy

[#] [back to top]

Property: tags:tag

tag - It associates a resource to a Tagging object.

Range:	Tagging tags:Tagging

[#] [back to top]

Property: tags:taggedWithTag

taggedWithTag - It indicates that the object has been tagged with the Tag Object.

Range:	Tag tags:Tag

[#] [back to top]

Property: ewe:timesUsed

timesUsed - Statistical property that stores the number of times a Service, Event or Action has been used. Thus, the value it stores corresponds to an snapshot of the Rule Database, for instance, at the time of the extraction from the origian Task Automation Service.

Domain:	Rule
Range:	xsd:int

[#] [back to top]

Property: ewe:triggeredBy

triggeredBy - Connects the left part of the ECA rule, the Event that triggers the rule.

Domain:	Rule
Range:	Event

[#] [back to top]

Property: ewe:uses

uses - Points out a Service that is used in the definition of the AppRule, either in the left part -the event- or the right part -the action.

Domain:	Rule
Range:	Channel
Has inverse property	isUsedIn

[#] [back to top]

A. Change Log

2013 - 05 - 20

- First version of the document

B. Acknowledgments

This documentation has been generated automatically from the most recent ontology specification in OWL using a python script called SpecGen. The style formatting has been inspired on FOAF specification.

Special thanks for support with ontology creation and research to: Prof. Carlos A. Iglesias and members of the GSI Group of DIT department of Universidad Politécnica de Madrid.

This work has been funded by the Spanish *Ministerio de Economía y Competitividad* through the Calista project (TEC2012-32457).

Glossary

BDI Belief-Desire-Intention

CEP Complex Event Processing

dcterms DCMI Metadata Terms

DOAP Description Of A Project

ECA Event–Condition–Action

EPTS Event Processing Technology Society

EWE Evented WEb ontology

FOAF Friend of a Friend ontology

GN GeoNames Ontology

LOD Linked Open Data

LOV Linked Open Vocabularies

MAIA Modular Architecture for Intelligent Agents

MOM Message Oriented Middleware

NLI Natural Language Interface

NLP Natural Language Processing

N3 Notation3

OWL Ontology Web Language

PA Personal Assistants

PRR Production Rule Representation

RDF Resource Description Framework

RIF	Rule Interchange Format
SIOC	Semantically-Interlinked Online Communities
SKOS	Simple Knowledge Organization System
SPARQL	SPARQL Protocol and RDF Query Language
SPIN	SPARQL Inferencing Notation
SSN	Semantic Sensor Network ontology
SWRL	Semantic Web Rule Language
TAGS	Tag Ontology
TAS	Task Automation Service
WEF	WSDM Event Format
WoT	Web of Things